Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROYECTO FINAL DE CARRERA

Estimación eficiente de la entropía y generación de anomalías para herramientas de monitorización de red

(Efficient estimation of entropy and anomaly generation for network monitoring tools)

Estudios: Ingeniería de Telecomunicaciones

Autor: Carlos Alejandro López Cruces

Director: Pere Barlet i Ros

Año: *2010-2011*

UPC

telecom
BCN

Índice

Agradecimientos 1

Resumen del Proyecto 3

Resum del Projecte 5

Abstract 7

1 Introducción 9

2 Objetivos 13

3 Estimación de la entropía 15

3.1 Introducción 15

3.2 Concepto de entropía 16

3.3 Streaming Algorithm 18

3.4 Sieving algorithm 19

3.5 Evaluación de los algoritmos 20

3.5.1 Error relativo 24

3.5.2 Memoria requerida 25

3.5.3 Coste computacional 29

3.6 Conclusiones 31

4 Generador de anomalías 33

4.1 Introducción 33
4.2 Anomalías en redes 34
4.3 Captura de tráfico de red 35
4.3.1 Libpcap 36
4.3.2 Netflow..38
4.3.3 DAG Cards Extensible Record Format – ERF...40
4.4 Flame 41
4.5 Desarrollo del generador de anomalías..42
4.5.1 Descripción 42
4.5.2 Modelado de anomalías...45
4.5.3 Generación de una anomalía..46
4.6 Conclusiones 48

5 Load Shedding y uso de la entropía.. 49

5.1 Introducción 49
5.2 CoMo 50
5.2.1 Estructura del sistema...50
5.2.2 Procesos núcleo 51
5.2.3 Módulos 52
5.2.4 Interfaz gráfica: CoMo-Live! 52
5.3 Gestión de recursos en CoMo 53
5.3.1 Introducción 53
5.3.2 Subsistema de predicción y control de carga 54
5.4 Evaluación de las nuevas features basadas en la entropía 55
5.4.1 Error de predicción 57
5.4.2 Consumo de las nuevas features 59
5.4.3 Comportamiento de las features en presencia de anomalías...........................60

5.5 Conclusiones 68

6 Planificación y estudio económico 71

6.1 Planificación 71

6.2 Estudio económico 74

6.2.1 Presupuesto de recursos humanos....74

6.2.2 Presupuesto de equipamiento informático 74

6.2.3 Presupuesto general 75

7 Conclusiones finales y trabajos futuros 77

8 Bibliografía 81

9 Índice de figuras.... 83

10 Índice de tablas 85

A Algoritmos de estimación de la entropía 87

A.1 Streaming Algorithm....87

A.2 Sieving Algorithm....88

B Modelos de anomalías de ejemplo.... 89

B.1 DDoS 89

B.2 NetScan 92

B.3 Mínimo y máximo de entropía 94

Agradecimientos

La presentación de este proyecto supone la culminación de muchos años de estudio y esfuerzo que finalizan en esta memoria y su presentación. A lo largo de mi vida ha habido muchas personas que han influido directa o indirectamente en el camino que me ha llevado hasta aquí.

A los primeros que me gustaría dar las gracias es a mis padres y mi familia, que se han esforzado durante muchos años para que yo pudiese recibir la mejor educación posible. Espero que sepan que todo ese esfuerzo no ha sido en vano.

A cada uno de los profesores que me han ido formando poco a poco desde que empecé en la escuela hasta mis últimos profesores de la universidad, y que me han dado las bases para poder terminar esta carrera. Entre ellos, darle las gracias a mi tutor por su paciencia respecto a mis largos silencios durante la realización del proyecto.

A mis amigos y compañeros de universidad, sin los cuales no habría logrado llegar hasta el final de la carrera, y que me han apoyado y presionado para poder acabar este proyecto y no dejarlo a medias.

Y sobretodo a Núria que siempre ha estado a mi lado apoyándome y animándome, y sin la cual nada de esto tendría sentido.

Resumen del Proyecto

El objetivo del presente proyecto es mejorar el sistema de gestión de recursos de la herramienta de monitorización de red CoMo en presencia de amenazas o anomalías en el tráfico de red. Para ello se ha considerado utilizar el valor de la entropía como indicador para el sistema de gestión de recursos de CoMo, basado en las técnicas de Load Shedding. Para alcanzar este objetivo se han desarrollado dos algoritmos de estimación eficiente de la entropía para obtener mayor información del tráfico y poder ser utilizados en enlaces de alta velocidad; se ha desarrollado un generador de anomalías para poder realizar simulaciones de tráfico con anomalías; y finalmente se ha evaluado si el uso de la entropía mejora el sistema de gestión de recursos de CoMo, tanto en ausencia como en presencia de anomalías.

UPC telecom BCN

Resum del Projecte

El present projecte té com objectiu millorar el sistema de gestió de recursos de l'eina de monitoratge de xarxa CoMo en presencia d'amenaces o anomalies en el tràfic d'una xarxa. Per això s'ha considerat utilitzar el valor de l'entropia com indicador pel sistema de gestió de recursos de CoMo i que està basat en les tècniques de Load Shedding. Per aconseguir aquest objectiu s'han desenvolupat dos algoritmes d'estimació eficient de l'entropia per obtenir més informació del tràfic i poder ser utilitzats en enllaços d'alta velocitat; s'ha desenvolupat un generador d'anomalies per realitzar simulacions de tràfic amb anomalies; i finalment, s'ha avaluat si l'ús de l'entropia millora el sistema de gestió de recursos de CoMo, tant en absència d'anomalies com en la seva presencia.

Abstract

The aim of this project is to improve the resource management system of the CoMo network monitoring tool in the presence of threats or traffic anomalies. To achieve this objective, the use of the entropy value is considered as an indicator for Load Shedding-based resource management system. For this reason, two algorithms for efficient entropy estimation have been developed to obtain more information of the traffic; an anomaly generator has been developed to be able to simulate network traffic with anomalies; and the use of the entropy in the resource management system has been evaluated to check if the system improves, in the lack of anomalies as well as in their presence.

Capítulo 1

Introducción

Desde su creación en un entorno militar, Internet ha supuesto una revolución sin precedentes en el mundo de las comunicaciones. Su capacidad para propagar información a nivel mundial de forma instantánea y de potenciar la colaboración e interacción entre individuos, instituciones y empresas ha permitido el desarrollo de multitud de servicios inconcebibles hasta su aparición. La necesidad de garantizar y mejorar el servicio que ofrece Internet junto con el aumento de las actividades delictivas tales como la distribución de virus, fraude electrónico o correo no deseado ha ocasionado la aparición de sistemas que permiten conocer el estado y comportamiento de la red.

Para ello, la monitorización del tráfico de red es una parte muy importante del estudio, de la gestión y de la seguridad de las redes actuales. Básicamente consiste en la captura y análisis de la información que transporta dicha red. Este análisis permite estudiar el comportamiento tanto de la red como de los protocolos utilizados, detectar y resolver errores o fallos que puedan aparecer o dimensionar de forma óptima los recursos de la red, entre muchas otras aplicaciones.

Un ejemplo de este tipo de herramientas es CoMo (Continuous Monitoring), la cual pretende ser un sistema de monitorización pasivo de propósito general. Está desarrollado por Intel Research, con la colaboración de otros centros, entre los que se incluye el Centro de Comunicaciones Avanzadas de Banda Ancha (CCABA) de la Universidad Politécnica de Cataluña (UPC).

CoMo es el elemento central de una infraestructura abierta de monitorización de tráfico de redes que permite a investigadores y operadores de red procesar y compartir de forma muy sencilla estadísticas del tráfico. El sistema permite calcular cualquier métrica genérica sobre el tráfico capturado, proporciona privacidad y seguridad tanto al propietario de la red como a los usuarios, y es robusto frente a patrones de tráfico anómalos. Dispone de una arquitectura modular que permite añadir nuevas funcionalidades y calcular nuevas métricas del tráfico de forma sencilla mediante la inserción de nuevos módulos desarrollados en C. Además el usuario puede analizar

tanto tráfico en tiempo real como tráfico registrado en diferentes formatos (PCAP, ERF, Netflow,...) para obtener cualquier información o métrica que proporcionen los módulos desarrollados.

Un sistema de monitorización de red debe ser capaz de hacer frente a los inevitables efectos producidos por situaciones de sobrecarga, debido tanto a grandes volúmenes de tráfico, a altas tasas de transmisión de datos o a la propia naturaleza a ráfagas del tráfico de red durante su funcionamiento normal. Las técnicas de *Load Shedding* permiten reducir la carga de un sistema cuando está trabajando en unas condiciones límite mediante el descarte o muestreo de una cierta cantidad de paquetes del tráfico de entrada.

Para realizar este muestreo, CoMo basa su decisión en el análisis volumétrico de diferentes características que describen las propiedades del tráfico de la red. Entre los cálculos que actualmente realiza el módulo de *Load Shedding* de CoMo se encuentran, el número de paquetes, el número de direcciones IP origen y destino únicas, el número de protocolos diferentes, entre otros.

Entendiendo la importancia de estos mecanismos de gestión de recursos, lo que se pretende en este proyecto es mejorar el sistema de Load Shedding de CoMo mediante el uso de nuevas métricas basadas en el cálculo de la entropía. En principio, estos nuevos valores permitirán obtener información del tráfico de entrada que hasta ahora podía pasar desapercibida con las métricas que se utilizan actualmente en CoMo. Para realizar este cálculo de la entropía se han desarrollado dos algoritmos encargados de realizar una estimación de su valor de la forma más eficiente posible, para facilitar el ser utilizados en enlaces de alta velocidad.

Por otra parte, la necesidad de evaluar sistemas de monitorización frente a ataques y anomalías requiere de capturas de tráfico de red que los contengan. Una buena aproximación para obtener este tipo de tráfico es el de generarlo uno mismo, lo que permite tener un buen conocimiento de la anomalía presente en la captura de tráfico. Para ello, en este proyecto se ha desarrollado una herramienta capaz de añadir de una forma rápida y sencilla anomalías sobre una traza de tráfico previamente capturada.

Por último, se ha comprobado el impacto de estas nuevas características basadas en la entropía en el proceso de predicción que lleva a cargo el modulo de gestión de recursos. De esta forma, y aprovechando el generador de anomalías desarrollado, se ha utilizado la estimación de la entropía tanto en presencia como en ausencia de anomalías para analizar si realmente el trabajo desarrollado consigue mejorar el sistema de gestión de recurso de CoMo o si por el contrario la entropía nos aportará información redundante.

Los resultados obtenidos en el presente proyecto son los siguientes: los algoritmos de estimación de entropía que se han desarrollado han dado buenos resultados, realizando estimaciones con poco error y de forma eficiente; el generador de anomalías desarrollado cumple con su propósito y ya está siendo utilizado por alumnos de posgrado del departamento de arquitectura de computadores; y por último se ha comprobado que el uso de la entropía mejora el módulo de gestión de recursos de CoMo, haciéndolo más robusto frente a determinadas anomalías que actualmente no es capaz de detectar.

Con el fin de exponer todos los procedimientos llevados a cabo, la presente memoria está distribuida de la siguiente forma: Tras este primer capítulo de introducción, en el capítulo 2 se describen los objetivos del proyecto, a la vez que se define la metodología utilizada para alcanzarlos. En el capítulo 3 se presentan dos algoritmos capaces de realizar una estimación de la entropía y se evalúan respecto al cálculo exacto de la entropía, obteniendo su error relativo, consumo de memoria y el número de ciclos de CPU utilizados. En el capítulo 4 se describe el desarrollo y funcionamiento del generador de anomalías. En el capítulo 5 se muestran los resultados obtenidos al utilizar la entropía dentro de los algoritmos de Load Shedding de la herramienta CoMo en presencia y ausencia de anomalías. En el capítulo 6 se presenta la planificación de este proyecto, y por último en el capítulo 7 se presentan las conclusiones finales y los trabajos futuros a realizar.

Capítulo 2

Objetivos

Este proyecto de final de carrera se divide en tres bloques los cuales pretenden alcanzar los siguientes objetivos.

- Desarrollar un algoritmo capaz de obtener el valor estimado de la entropía de forma eficiente.
- Desarrollar una aplicación capaz de introducir anomalías en una traza de tráfico de red previamente capturada, haciendo uso de las librerías libpcap tanto para la lectura como para la escritura de datos. Además, el desarrollo de nuevas anomalías debe ser rápido y simple.
- Comprobar si el módulo encargado de la gestión de recursos de CoMo mejora al utilizar la entropía como elemento de decisión para realizar las predicciones de consumo de CPU, tanto en ausencia como en presencia de anomalías.

Cada uno de estos tres objetivos se corresponde con los capítulos 3, 4 y 5 de esta memoria respectivamente, en los cuales se presentan los conocimientos teóricos necesarios y se exponen todos los procedimientos realizados para poder cumplir con los objetivos citados.

Para alcanzar el primer objetivo se han evaluado dos algoritmos basados en el artículo *Data Streaming Algorithms for Estimating Entropy of Network Traffic* [1] realizado por Ashwin Lall et. al. Estos dos algoritmos han sido comparados para ver cual de los dos ofrece un mejor comportamiento en los siguientes términos:

- Error relativo frente al cálculo exacto de la entropía.
- Espacio en memoria necesario para estimar el valor de la entropía frente al cálculo exacto de la entropía.
- Ciclos de CPU necesarios para realizar la estimación frente al cálculo exacto de la entropía.

Para cumplir el segundo objetivo, el desarrollo del generador de anomalías se ha basado en la aplicación FLAME la cual es una herramienta de modelado de anomalías en formato Netflow realizado por miembros del Instituto Federal Suizo de tecnología de Zurich. La nueva herramienta consiste en una ampliación de FLAME para permitir la lectura y escritura de trazas en formato Libpcap.

Para conseguir el tercer objetivo se ha escogido el algoritmo que obtiene mejores resultados para la estimación de la entropía y se ha añadido a la herramienta de monitorización CoMo dentro del módulo de *Load Shedding*. Además se ha comprobado el error de predicción del consumo de CPU que realiza este módulo cuando utiliza estos nuevas métricas para comprobar si el uso de la entropía consiguen mejora el sistema de gestión de recursos de CoMo.

Capítulo 3

Estimación de la entropía

3.1 Introducción

El término entropía ha sido tomado prestado de la termodinámica para designar la cantidad de información media que contiene una fuente de datos. En 1948 Claude Shannon publicó *A Mathematical Theory of Communication* [2] en el cual se introducen los conceptos de información y entropía de una fuente de datos. Intuitivamente la entropía es una medida de la diversidad o aleatoriedad de un flujo de datos cuyo valor está acotado por el mínimo igual a 0 cuando todos los mensajes son iguales y el máximo valor igual a $log_a(m)$ el número cuando todos los mensajes son diferentes (con a la base de cálculo de la entropía y m el número total de mensajes).

El uso de la entropía de la distribución del tráfico se ha demostrado que ayuda en un amplía variedad de aplicaciones de monitorización de red, tales como detección de anomalías, agrupación para revelar patrones o clasificación del tráfico. Sin embargo, el uso de algoritmos que realicen un cálculo exacto de la entropía pueden no ser aplicables en conexiones de alta velocidad debido a requisitos tales como el consumo de CPU o el uso de memoria.

A continuación se presentan dos algoritmos que permiten estimar el valor de la entropía de forma eficiente, es decir, intentan utilizar una menor cantidad de recursos respecto al cálculo del valor exacto, pero manteniendo un error relativo bajo en la estimación. En el caso particular de estos algoritmos, cuanto más eficientes sean en cuanto a uso de recursos, peores estimaciones de la entropía obtendremos; por lo tanto existirá un compromiso entre eficiencia y calidad del estimador en función de los diferentes parámetros de diseño de los estimadores.

Estos algoritmos se han desarrollado y probado sobre la aplicación de monitorización de tráfico CoMo estimando la entropía de diferentes propiedades del tráfico que captura esta herramienta. Para ello, se han añadido los cálculos necesarios dentro del módulo encargado de la gestión de

recursos de la aplicación. Una descripción más detallada de este módulo y de la herramienta CoMo se puede encontrar en el capítulo 5 de este documento.

Por último, cabe destacar que para poder comparar ambos algoritmos se ha desarrollado el algoritmo que calcula el valor exacto de la entropía y se han enfrentado sus resultados con los proporcionados por los dos algoritmos propuestos. Los criterios de comparación que se han utilizado para realizar esta evaluación son los siguientes:

- Error relativo frente al valor exacto de la entropía. Al tratarse de un estimador, ambos algoritmos tendrán un error asociado.
- Espacio en memoria. Se pretende desarrollar un algoritmo que necesite el mínimo espacio en memoria necesario manteniendo un error relativo bajo.
- Número de ciclos de CPU. Determina la velocidad del algoritmo.

3.2 Concepto de entropía

Etimológicamente la palabra entropía procede del griego, de *em* (en - en, sobre, cerca de...) y *sqopg* (tropêe - mudanza, giro, alternativa, cambio, evolución...). La primera vez que se utilizó este término fue en 1850 por el físico alemán Rudolf Julius Emmanuel Clausius tratando de hacer más claro el significado de la segunda ley de la termodinámica. A pesar de ello, la entropía es un concepto que se utiliza tanto en termodinámica como en mecánica estadística y en teoría de la información, que se concibe como una "medida del desorden" o la "peculiaridad de ciertas combinaciones".

Como la entropía puede ser considerada una medida de la incertidumbre, y la información tiene que ver con cualquier proceso que permite acotar, reducir o eliminar la incertidumbre, resulta que el concepto de información y el de entropía están ampliamente relacionados entre sí aunque se tardó años en el desarrollo de la mecánica estadística y la teoría de la información para hacer esto aparente. El encargado de adaptar esta palabra al mundo de las comunicaciones fue Claude Shannon en 1948 cuando publicó *A Mathematical Theory of Communication* [2] en el que se introducen los conceptos de información y entropía de una fuente de datos.

En un proceso sujeto a incertidumbre es común usar la teoría de la probabilidad para representar dicho proceso. En concreto, si el resultado de un proceso es un conjunto de posibles resultados, podemos definir una variable aleatoria X que puede tomar como posibles valores los resultados de dicho proceso en distintas repeticiones. Dichos resultados pueden ser equiprobables o ser unos más frecuentes que otros, y es precisamente la distribución de probabilidad de los valores

de X la que describe con qué frecuencia aparecerá cada uno de los posibles resultados del proceso sujeto a incertidumbre.

El cálculo general de la entropía se realiza mediante la siguiente expresión:

$$H(x) = E\{I(X) = \sum_{i=1}^{N} p(x_i) \times \log_n \left(\frac{1}{p(x_i)} \right) = -\sum_{i=1}^{N} p(x_i) \times \log_n \left(p(x_i) \right) \qquad 3.1$$

Donde *X* es una variable aleatoria con *N* posibles sucesos y $p(x_i)$ es la probabilidad del suceso i-ésimo. En el caso del cálculo de la entropía de la distribución de alguna característica del tráfico de red, no se dispone de las probabilidades de cada una de las diferentes variables, por lo que es necesario utilizar la siguiente expresión para calcular su valor exacto:

$$H = -\sum_{i=1}^{N} \frac{m_i}{m} \times \log_2 \left(\frac{m_i}{m} \right) = \log_2(m) - \frac{1}{m} \sum_{i=1}^{N} m_i \times \log_2(m_i) = \log_2(m) - \frac{1}{m} S \qquad 3.2$$

Con:

$$S = \sum_{i=1}^{N} m_i \times \log_2(m_i) \qquad 3.3$$

Donde *N* es el número total de posibles elementos, m_i es el número de veces que aparece el elemento i-ésimo y *m* es el número total de elementos, en nuestro caso el número total de paquetes. Por ejemplo, para el caso de querer obtener información sobre como está distribuido el tráfico de red en función del puerto destino se calcularía la entropía de esta característica utilizando los valores siguientes: N igual a 65535 que corresponde al número máximo de puertos posibles, m_i sería un contador para cada puerto destino y *m* correspondería al número total de paquetes capturados.

Por último, el valor de la entropía cumple las siguientes propiedades:

- $0 \leq H \leq \log_2(m)$, es decir, la entropía H esta acotada inferior y superiormente.
- Dado un procesos con posibles resultados $\{A_1,..,A_n\}$ con probabilidades relativas $p_1,...,p_n$, la función $H(p_1,\ldots,p_n)$ es máxima en el caso de que $p_1 = \ldots = p_n = 1/n$.
- Dado un procesos con posibles resultados $\{A_1,..,A_n\}$ con probabilidades relativas $p_1,...,p_n$ la función $H(p_1,\ldots,p_n)$ es nula en el caso de que $p_i = 0$ para todo *i* excepto un caso.

3.3 Streaming Algorithm

Este algoritmo está basado en el trabajo realizado por Ashwin Lall et al [1]. La idea principal del algoritmo es que el cálculo de S -definido en el apartado anterior- es estructuralmente similar a la estimación de los momentos de frecuencia cuya solución se plantea en [3]. La ventaja de esta técnica es que nos ofrece una estimación no sesgada de la entropía con un uso reducido de espacio en memoria y un buen rendimiento.

Este algoritmo realiza una aproximación (ε,δ) de S lo que implica que la estimación tiene un error relativo de ε como máximo con una probabilidad de por lo menos 1- δ, es decir, Pr(|X-X'| ≤ εX) ≥ 1-δ donde X y X' son el valor real y el estimado respectivamente.

Conceptualmente, este primer algoritmo se puede dividir en tres fases. En la primera fase se selecciona de forma aleatoria una serie de posiciones dentro del flujo de datos. Estas posiciones definen el conjunto de contadores que el algoritmo rastreará a lo largo de la segunda fase. En esta segunda fase, o fase en tiempo real, se mantendrá un seguimiento del número de veces que aparecen los elementos que se ubican en las posiciones seleccionadas en la primera fase. Por cada elemento seleccionado se almacenará un contador exacto del número de posteriores apariciones de ese elemento por lo que puede crearse más de un contador por elemento. Por ejemplo, si la posición k en el flujo de datos ha sido seleccionada, se guardará un contador con el número de veces que aparece ese elemento desde la posición k hasta el final del flujo de datos. En la tercera fase, el algoritmo utiliza los diferentes contadores que se han almacenado para obtener una estimación insesgada de S y cuyo error relativo debe ser bajo para finalmente calcular el valor normalizado de la entropía.

A continuación se presenta una descripción más detalla de las diferentes fases que se llevan a cabo para calcular el valor de S y consecuentemente el valor de la entropía.

- Fase de pre-procesado: Se seleccionan de forma aleatoria las posiciones de los paquetes de los cuales se van a mantener contadores. El número total de contadores viene determinado por el producto de las variables *z* y *g* que se calculan a partir de ε y δ según las siguientes ecuaciones:
  - $z = 32 \times \log_2\left(\frac{m}{\varepsilon^2}\right)$ 3.4
  - $g = 2 \times \log_2\left(\frac{1}{\delta}\right)$ 3.5

- Fase en tiempo real: Por cada paquete de la traza de tráfico se incrementan todos los contadores existentes. Además si éste ha sido seleccionado en la fase de pre-procesado, se creará un nuevo contador inicializado al valor 1.
- Fase de post-procesado: Por cada contador que se ha almacenado se calcula una estimación de S. Este estimador es no sesgado y se calcula mediante la siguiente fórmula:

$$X = m(c\log_2(c) - (c-1)\log_2(c-1)) \qquad 3.6$$

Una vez calculados estos $z * g$ estimadores insesgados se dividen en g grupos los cuales contiene cada uno z estimadores. Por cada grupo se calcula el valor medio y posteriormente se obtiene el valor de S a partir de la mediana de todos esos valores medios.

Finalmente a partir del valor de S se calcula el valor de H normalizado al valor máximo de la entropía mediante la siguiente expresión:

$$H = \frac{\log_2(m) - \frac{1}{m}S}{\log_2(m)} = 1 - \frac{S}{m \times \log_2(m)} \qquad 3.7$$

Una característica que se puede apreciar de este algoritmo es que el número de contadores depende del logaritmo del total de paquetes del que se estima la entropía (Ecuación 3.5). En la práctica esto tiene unas fuertes implicaciones en la eficiencia del algoritmo que se verá reflejado en los resultados obtenidos.

El pseudocódigo del algoritmo de Streaming se puede ver en el anexo A.1.

3.4 Sieving algorithm

El algoritmo anterior proporciona una estimación de la entropía de la fuente de datos independientemente de la estructura subyacente de la información a analizar. En la práctica, el tráfico de red suele seguir unos patrones determinados; en particular, una aproximación sencilla es el considerar que la distribución del tráfico suele tener una clara diferenciación entre flujos grandes (elefantes) y flujos pequeños (ratones). Es decir, un número pequeño de *flujos elefante* contribuyen a una gran cantidad de tráfico, mientras que el resto se consideran *flujos ratón* ya que suelen aparecer con menor frecuencia.

En este segundo algoritmo, también basado en el trabajo realizado en [1], se hace uso de esta idea de separar el tráfico en elefantes y ratones estimando por separado la contribución de cada uno de ellos al valor de la entropía. Con ello se pretende mejorar la precisión de los resultados mientras se reduce el uso de recursos por parte de dicho algoritmo.

Para este algoritmo se ha modificado ligeramente el método de muestreo utilizado frente al algoritmo de Streaming. En lugar de hacer un cálculo previo de las posiciones a utilizar, como se hacía en el algoritmo anterior, en éste se muestrea cada posición en función de una pequeña probabilidad. Una vez un paquete ha sido muestreado se mantiene un contador con el número exacto de veces que aparece en la traza de tráfico. Si un elemento solo se ha muestreado en una ocasión, entonces se le considera un ratón y se estima el valor de S de todos los ratones utilizando el estimador presentado en el algoritmo anterior. Sin embargo, si un elemento se ha muestreado en más de una ocasión, se le considera elefante y se calcula el valor exacto de su contribución a S. El valor final de S que se utilizará para calcular el valor de la entropía será la suma de ambos valores.

Debido a que no existe una fase de pre-procesado, este algoritmo se divide en dos fases. A continuación se describe de forma detallada cada fase de la misma forma que se ha hecho con el algoritmo anterior:

- Fase en tiempo real: Por cada paquete se comprueba si debe ser muestreado generando un número aleatorio y comprobando si es menor que cierta umbral. Si es muestreado y ya existe un contador, se considera ese elemento como un *elefante*; si no existe, se crea un nuevo contador con la categoría de ratón. Si el elemento no es muestreado se actualizan los contadores para el paquete (si existen).
- Fase de post-procesado: Por cada *elefante* se calcula su contribución a *S* mediante el cálculo $S_e = \sum_i c_i \times \log_2(c_i)$. La contribución de los *ratones* se realiza del mismo modo que la fase de post procesado de algoritmo anterior, obteniendo S_m. Finalmente la contribución de los *elefantes* y los *ratones* a la estimación de *S* es igual con lo que $S = S_e + S_m$

  Finalmente, al igual que en el algoritmo anterior, se calcula el valor de H a partir de S pero normalizado al valor máximo de la entropía mediante la expresión final presentada en el apartado anterior.

El pseudocódigo de este algoritmo se puede ver en el anexo A.2.

3.5 Evaluación de los algoritmos

El desarrollo y evaluación de estos algoritmos se ha realizado sobre la herramienta de monitorización CoMo aprovechando que el módulo de gestión de recursos calcula una serie de propiedades del tráfico que lo captura -una descripción más detallada tanto del módulo de

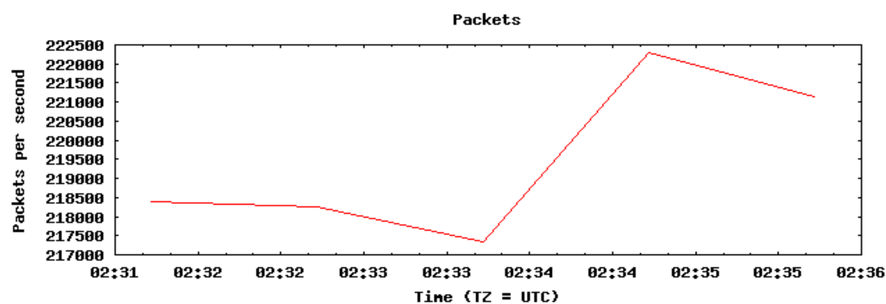
gestión de recursos como de la herramienta CoMo se puede encontrar en el capítulo 5 del presente proyecto-.

En particular, se ha estimado el valor de la entropía para diferentes características de este tráfico, es decir, por cada fragmento de tráfico capturado se estimará la entropía de las siguientes características:

- Dirección IP origen.
- Dirección IP destino.
- Dirección IP origen y destino.
- Protocolo y puerto origen.
- Protocolo y puerto destino.
- Protocolo, puerto y dirección Ip origen.
- Protocolo, puerto y dirección destino.
- Protocolo y puerto origen y destino.
- 5-tupla: Dirección IP origen y destino, puerto origen y destino y protocolo.
- Protocolo.

Para realizar la evaluación se han utilizado dos trazas de tráfico diferentes proporcionadas por el proyecto WITS (Waikato Internet Traffic Storage [4]). Ambas trazas provienen de un router en Indianápolis perteneciente a la red Abilene, el cual se encuentra dentro de un backbone de 10 gigabits por segundo. En concreto, la captura de tráfico se realiza sobre un enlace OC192c Packet-over-SONET en 2 períodos de tiempo diferentes. Las trazas tienen las siguientes características:

- Traza 1: Capturada el martes 8 de octubre de 2002 a las 21:31:43 UTC con una duración de 4 minutos la cual incluye 65 millones de paquetes o aproximadamente 20000 paquetes cada 100 milisegundos. La cantidad de paquetes a lo largo del tiempo i la velocidad del enlace en Mbps es la siguiente:



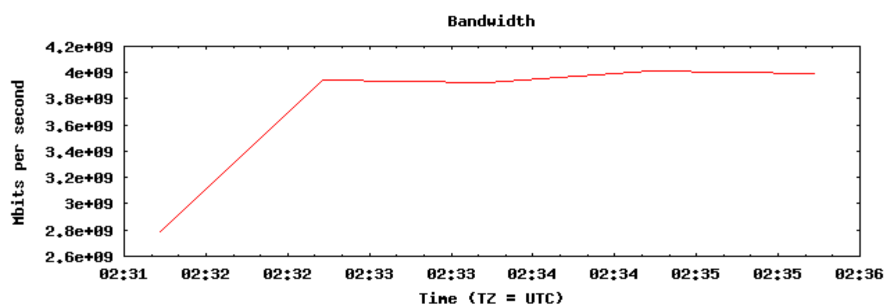


Figura 3-1 Propiedades de la Traza 1

- Traza 2: Capturada el miércoles 14 de agosto de 2002 a las 10:30:00 UTC con una duración de 9 minutos la cual incluye 47 millones de paquetes o aproximadamente 8000 paquetes cada 100 milisegundos. La cantidad de paquetes a lo largo del tiempo i la velocidad del enlace en Mbps es la siguiente:

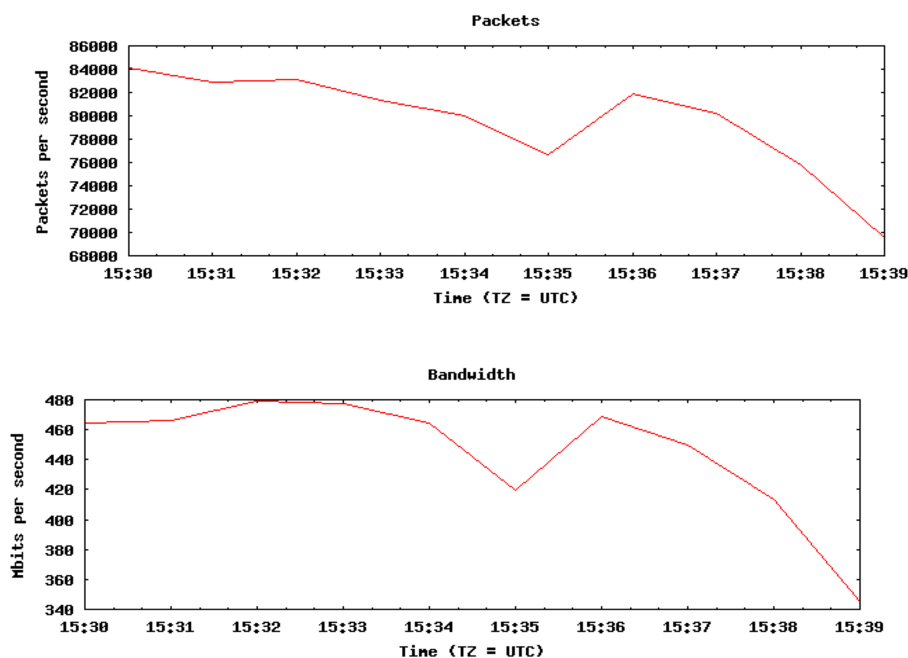


Figura 3-2 Propiedades de la Traza 2

Para estas trazas se ha calculado el valor exacto de la entropía y se ha estimado su valor mediante los dos algoritmos desarrollados. Además, la estimación de la entropía se realiza en fragmentos de 100 milisegundos, denominados *batch*, debido a que el módulo de gestión de recursos de CoMo utiliza estos fragmentos para realizar sus cálculos. Este hecho es importante por que limita el número máximo de paquetes que se utilizan para realizar el cálculo y, como se verá en los resultados obtenidos, la eficiencia de los algoritmos puede depender del tamaño del fragmento de tráfico que se utilice.

Cabe destacar también que ambos algoritmos disponen de varios parámetros que permiten modificar tanto la precisión como la eficiencia en la estimación de la entropía. Es por ello que para realizar su evaluación se han seleccionado diferentes valores para poder decidir la

configuración más óptima de cada algoritmo. En el caso del algoritmo Streaming, los valores utilizados para los parámetros ε y δ son los mostrados en la Tabla 3-1:

	ε	δ
Configuración 1	0.1	0.1
Configuración 2	0.1	0.2
Configuración 3	0.1	0.3
Configuración 4	0.1	0.5
Configuración 5	0.1	0.8

Tabla 3-1 Parámetros de simulación del algoritmo de Streming

En el caso del algoritmo Sieving se han utilizado los valores de la Tabla 3-2 para la probabilidad de muestreo de cada paquete:

	Probabilidad muestreo
Configuración 1	0.001
Configuración 2	0.01
Configuración 3	0.02
Configuración 4	0.03
Configuración 5	0.04
Configuración 6	0.05
Configuración 7	0.1

Tabla 3-2 Parámetros de simulación del algoritmo de Sieving

Por último, para poder comparar ambos algoritmos se ha desarrollado el algoritmo de cálculo exacto del valor de la entropía y se ha enfrentado su resultado con los proporcionados por los dos algoritmos propuestos. Los criterios de evaluación que se han utilizado son los siguientes:

- Error relativo frente al valor exacto de la entropía. Al tratarse de un estimador, ambos algoritmos tendrán un error asociado que deberá ser tan pequeño como sea posible para garantizar que las estimaciones que realizan sean fiables.
- Espacio en memoria. Se pretende desarrollar un algoritmo que necesite el mínimo espacio en memoria necesario manteniendo un error relativo bajo. Cuanto menor sea el error relativo, mayores serán los requisitos de espacio en memoria.
- Número de ciclos de CPU. Se pretende determinar la velocidad de cada algoritmo mediante la contabilización del número de ciclos de CPU que se requieren en cada fase en relación al cálculo del valor de la entropía exacta.

A continuación se presentan los resultados obtenidos para cada uno de los algoritmos y para las dos trazas evaluadas.

3.5.1 Error relativo

A partir del cálculo del valor real de la entropía y las estimaciones realizadas mediante los dos algoritmos propuestos se ha obtenido el error relativo de la estimación con la siguiente expresión:

$$E = \frac{\left| H - \tilde{H} \right|}{H} \qquad 3.8$$

Donde H es el valor real, $\tilde{H}$ es el valor estimado y E es el error relativo. Este criterio es fundamental para comprobar que ambos algoritmos son lo suficiente precisos como para considerar que los valores obtenidos de la entropía son válidos. Se ha considerado que el error relativo de las estimaciones debe ser de alrededor del 5% para considerar la estimación como válida.

A continuación se presentan los resultados obtenidos para ambos algoritmos y para las dos trazas utilizadas. Además, se ha realizado la evaluación para las 9 características definidas y los diferentes valores de los parámetros de configuración de cada algoritmo.

La Figura 3-3 muestra el error relativo medio en la estimación de la entropía del algoritmo de Streaming para los diferentes valores de ε y δ y para algunas de las diferentes features.

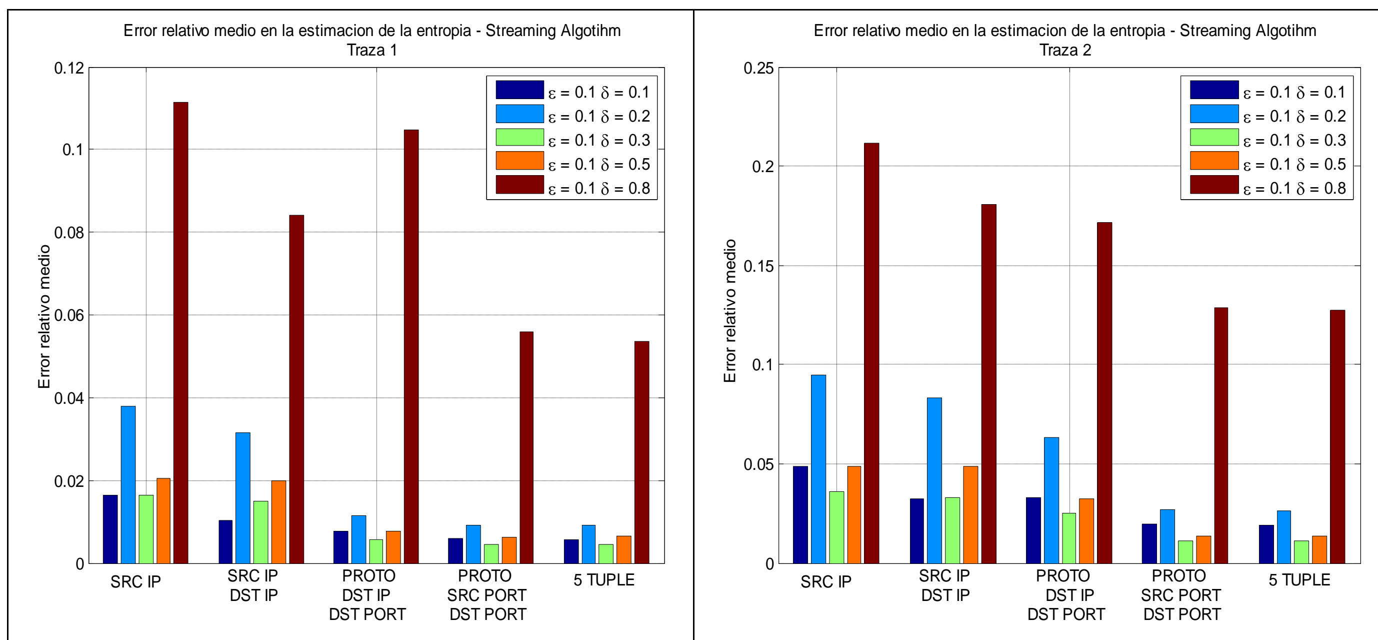


Figura 3-3 Error relativo en la estimación de la entropía del algoritmo de Streaming

Se puede observar que para configuraciones con un número de contadores muy bajo, ε=0,1 y δ=0,8, el error relativo es realmente alto siendo en algunos casos superior al 10%, debido a que

esta configuración teóricamente ofrece un 10% de error en el 20% de los casos. Sin embargo con en el resto de configuraciones el error se reduce, obteniendo valores inferiores al 5% en casi todas las configuraciones.

En la Figura 3-4 se pueden ver los resultados obtenidos para el algoritmo de Sieving y para las dos trazas de estudio. Al igual que el algoritmo anterior, se ha realizado la evaluación con diferentes configuraciones variando el valor de la probabilidad de muestreo desde 1 de cada mil paquetes hasta 1 de cada 10.

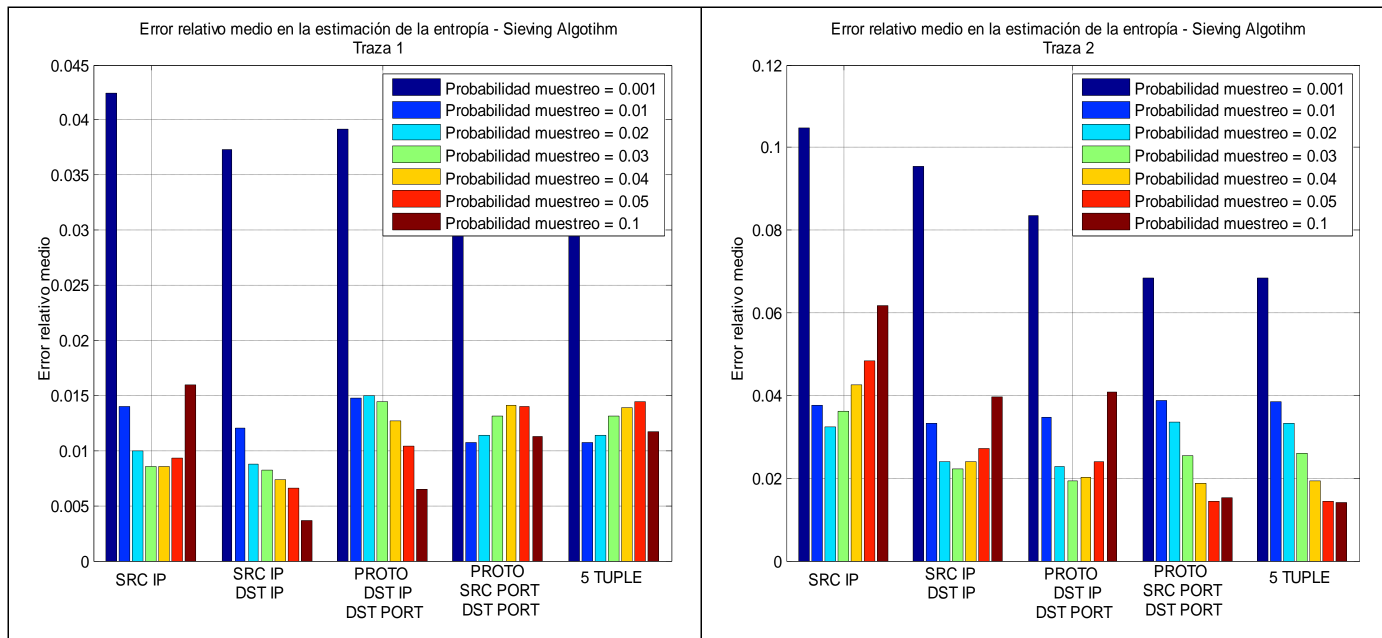


Figura 3-4 Error relativo en la estimación de la entropía del algoritmo de Sieving

Se puede observar en la figura anterior que el error relativo medio del algoritmo de Sieving es menor que para el algoritmo de Streaming, con unos valores que varían des del 1% hasta un 5%, según la probabilidad de muestreo y la característica que se considere, a excepción de cuando muestreamos uno de cada 1000 paquetes que incrementa considerablemente el error (valores entre el 4 y el 10%). Además, aunque no se muestra en la figura, también se ha calculado la desviación estándar de las estimaciones obteniendo está por debajo del 0.5%.

Con estos datos se puede decir que el algoritmo de Sieving es mejor que el de Streaming, pero se debe comprobar como de eficientes son ambos algoritmos al estimar la entropía para poder llegar a una conclusión definitiva. A continuación se realizará este análisis a partir del cálculo del espacio en memoria que utilizan ambos algoritmos y de la velocidad en su ejecución

3.5.2 Memoria requerida

Uno de los objetivos del algoritmo a desarrollar es que sea capaz de reducir considerablemente el espacio en memoria necesario para poder estimar el valor de la entropía. En la práctica, la

memoria requerida para su estimación vendrá determinada tanto por el número de contadores que se utilizarán para su cálculo como por el tamaño de cada uno de ellos. Sin embargo, el tamaño de los contadores de ambos algoritmos es el mismo, por lo que el mecanismo que utilizan para aumentar su eficiencia es el de reducir el número de estos contadores.

Los resultados presentados a continuación muestran la eficiencia de cada algoritmo representada por la siguiente ecuación: Contadores Valor Exacto

$$Eficiencia = \frac{N^{o}\ \text{Contadores} \quad \text{Valor} \quad \text{Exacto}}{N^{o}\ \text{Contadores} \quad \text{Algoritmo}} \qquad 3.9$$

Al ser el número de contadores de cada algoritmo menor que el número de contadores necesario para el cálculo exacto de la entropía, el valor obtenido pretende representar la eficiencia de estos algoritmos. Por ejemplo, si para el cálculo del valor de la entropía se necesitan 100 contadores y para su estimación se necesitan 50, esto implica que el algoritmo de Sieving es el doble de eficiente al utilizar la mitad de contadores para hacer la estimación

La Figura 3-5 muestra los resultados obtenidos para el algoritmo de Streaming y para las dos trazas de tráfico analizadas en función de los dos parámetros de configuración de este algoritmo.

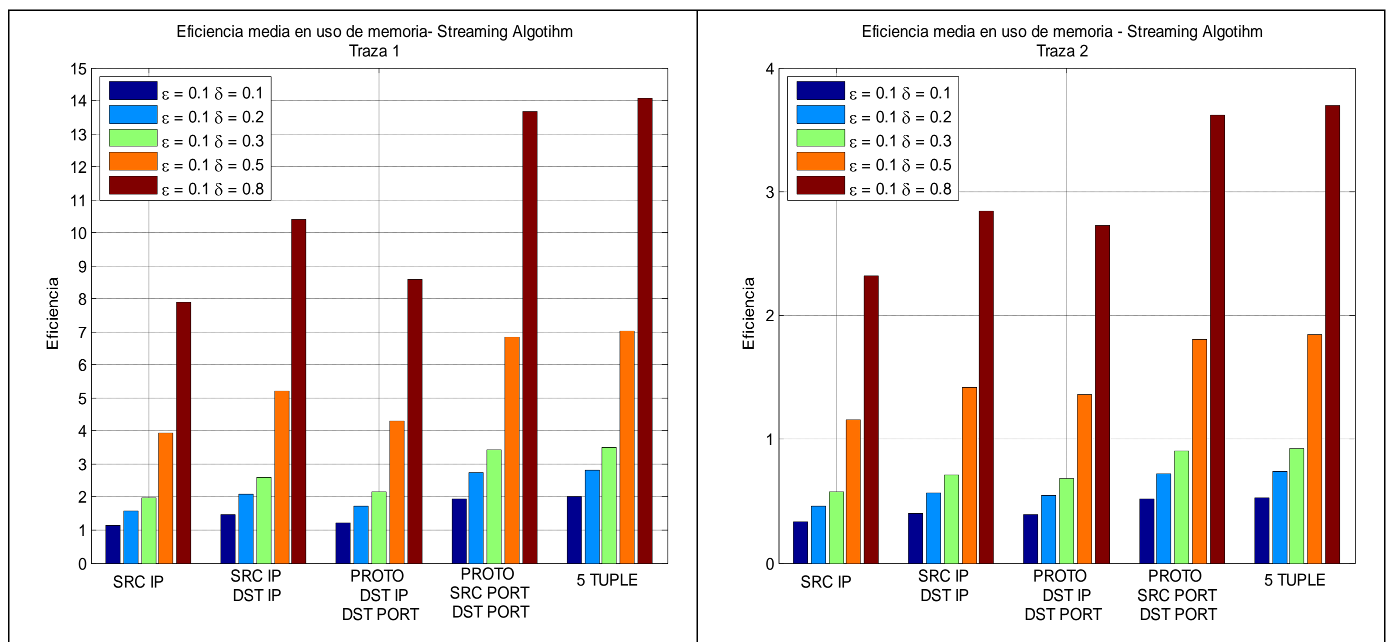


Figura 3-5 Eficiencia media del algoritmo de Streaming respecto al cálculo exacto

De la figura anterior se puede concluir que el algoritmo de Streaming es bastante ineficiente ya que para determinadas configuraciones el número de contadores que se utilizan es comparable al del algoritmo de cálculo exacto de la entropía, e incluso para la segunda traza de tráfico se utiliza un mayor número de contadores para la estimación que para el cálculo exacto. Esto se debe principalmente a que el número de contadores es inversamente proporcional al tamaño del

fragmento de tráfico, hecho que se puede comprobar al ver que este algoritmo es más eficiente para la primera traza, con alrededor de 16000 paquetes por fragmento, que para la segunda, con alrededor de 8000.

Esta relación también se puede observar en la Ecuación 3.4 de la variable z la cual depende del logaritmo del número total de paquetes y determina el número total de contadores que se utilizan. La Figura 3-6 muestra esta relación entre paquetes y contadores donde se pueda observar que el número de contadores aumenta muy lentamente en comparación con el número de paquetes, lo que implica un aumento de la eficiencia al utilizar fragmentos de tráfico con más paquetes.

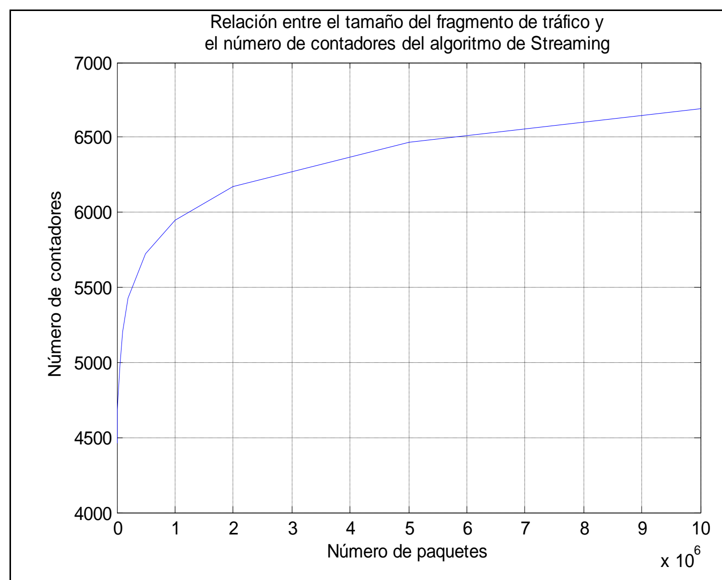


Figura 3-6 Relación entre el número de paquetes y el número de contadores para el algoritmo de Streaming

Por otra parte, la eficiencia del algoritmo de Sieving se muestra en la Figura 3-7 para las diferentes configuraciones que se han utilizado y para las dos trazas de estudio.

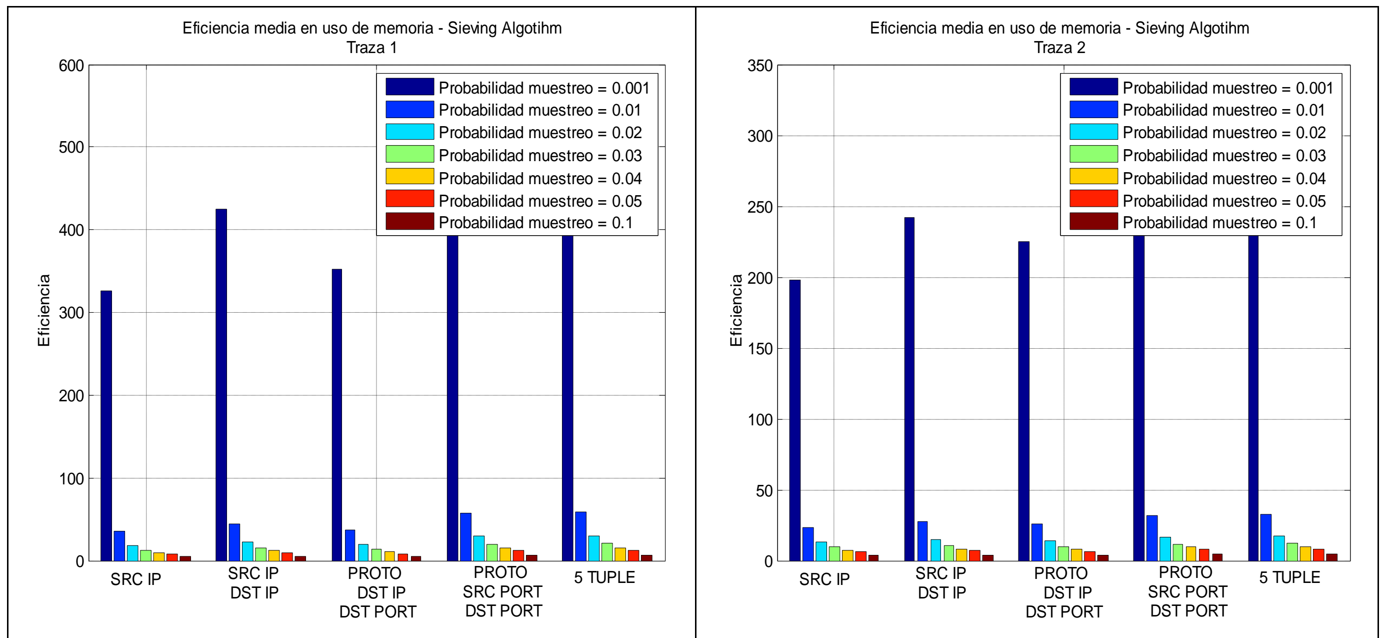


Figura 3-7 Eficiencia media del algoritmo de Sieving y respecto el cálculo exacto

En este caso, los resultados muestran que el algoritmo de Sieving es muy eficiente estimando la entropía, ya que como mínimo se han utilizado un 25% de contadores respecto al cálculo exacto de la entropía para la configuración menos eficiente (para la probabilidad de muestreo de 0.1 el valor de la eficiencia es superior a 4), por lo que supera con creces al algoritmo de Streaming.

Por último, en la presentación de esta evaluación se ha indicado que la eficiencia del algoritmo de Streaming aumenta cuanto mayor es el tamaño del fragmento de red utilizado. En particular, la Figura 3-8 muestra como evoluciona el error relativo medio y la eficiencia del algoritmo de Streaming en función de la duración del fragmento de datos del que se realiza la estimación, para el caso de las direcciones IP origen.

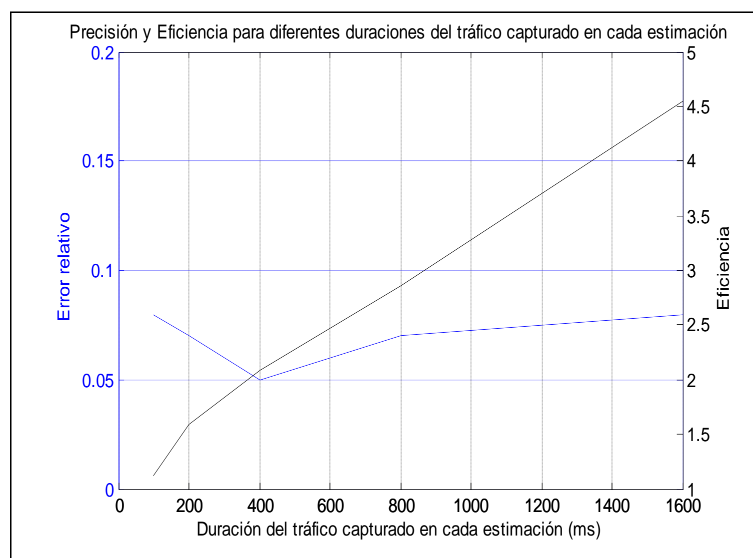


Figura 3-8 Error relativo y eficiencia del algoritmo de Streaming respecto a la duración del fragmento de captura al realizar la estimación

Los resultados muestran que la eficiencia del algoritmo aumenta considerablemente sin aumentar significativamente el error de las estimaciones, lo que implica que el algoritmo no es tan ineficiente como puede parecer con los primeros resultados obtenidos. Esto quiere decir que son necesarios un número mínimo de contadores para realizar estimaciones con un error máximo, independientemente del número de paquetes que se utilicen para realizar la estimación.

A pesar de ello, ha quedado demostrado que el algoritmo de Sieving es más eficiente independientemente del tamaño del fragmento de tráfico que se utilice.

3.5.3 Coste computacional

Otra forma de analizar la eficiencia de ambos algoritmos es comparando la velocidad de ejecución frente al número de ciclos necesarios para calcular el valor real de la entropía. Como ya se ha visto en la descripción, ambos algoritmos disponen de una fase de ejecución en tiempo real en la que se actualizan los valores de los contadores y otra de post-procesado en la que se estima el valor de la entropía. Es por ello que se ha separado el análisis en estas dos fases para finalmente sacar una conclusión.

Los resultados se presentan como la relación entre el número de ciclos que utiliza cada algoritmo y el número de ciclos necesarios para calcular el valor exacto de la entropía para cada una de las fases. Para el cómputo del número de ciclos de CPU se han utilizado las mismas funciones que utiliza el módulo de gestión de recursos de CoMo en el que ya se realiza una tarea similar.

A continuación se muestran los resultados obtenidos para el algoritmo de Streaming y para las dos trazas de tráfico analizadas en función de los dos parámetros de configuración de este algoritmo.

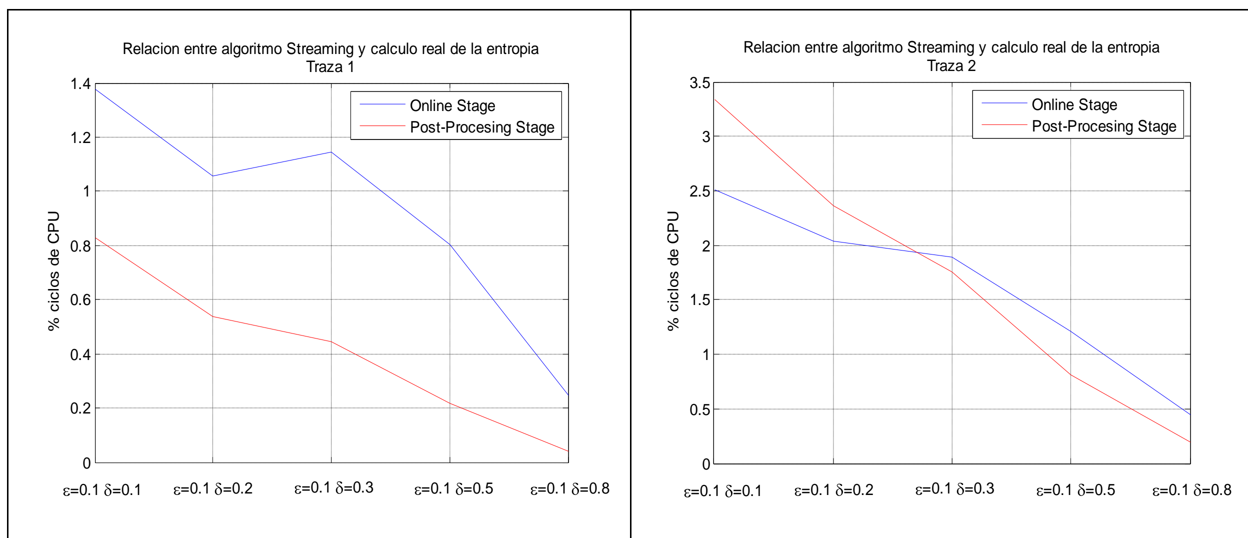


Figura 3-9 Porcentaje de ciclos de CPU para la estimación de la entropía - Streaming

La Figura 3-9 muestra que el algoritmo de Streaming requiere un mayor número de ciclos de CPU que el cálculo exacto de la entropía debido a que existe una relación directa entre el coste del algoritmo y el número de contadores que se han utilizado. Como ya se ha visto en el apartado anterior, este algoritmo es muy ineficiente para fragmentos pequeños de tráfico y este hecho se ve claramente reflejado en el número de ciclos de CPU que utiliza.

La Figura 3-10 muestra los resultados obtenidos con el algoritmo de Sieving para diferentes valores de la probabilidad de muestreo, tanto para la fase en tiempo real y para la fase de post-procesado.

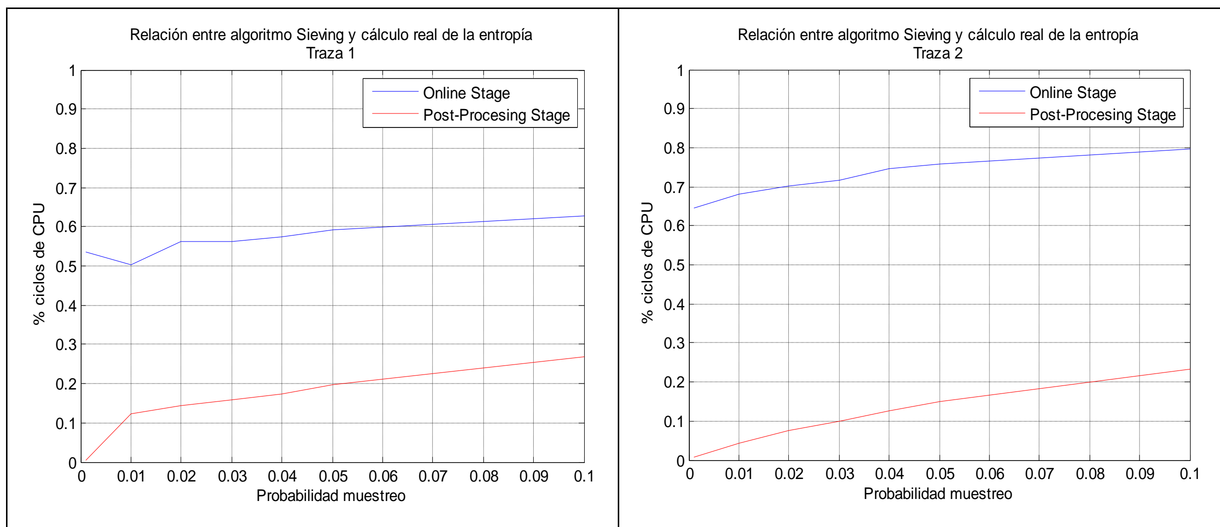


Figura 3-10 Porcentaje de ciclos de CPU para la estimación de la entropía - Sieving

De la misma forma que el análisis de memoria requerida, el algoritmo de Sieving es más eficiente que el cálculo exacto de la entropía, sobretodo en la fase de post-procesado. Además, de las figuras anteriores se pueden deducir las siguientes conclusiones:

- Se observa una reducción de ciclos en la fase en tiempo real para la traza 1 de alrededor del 50% y para la traza 2 del 30%. Mejorar estos resultados es difícil debido a que tanto el cálculo exacto de la entropía como la estimación de ambos algoritmos actualizan el valor de los contadores en función del contenido de cada paquete. Además, los resultados obtenidos muestran que no hay tanta dependencia con la configuración utilizada como pasaba con el algoritmo de Streaming.
- Al igual que en la fase de tiempo real, el tiempo de ejecución de la fase de post-procesado está relacionado con el número de contadores que se almacenan. Por este motivo las figuras anteriores muestran que el algoritmo de Sieving es entre 4 y 7 veces más rápido que el cálculo exacto de la entropía en la fase de post-procesado con las configuraciones utilizadas.

3.6 Conclusiones

Tras evaluar los resultados obtenidos podemos concluir que ambos algoritmos realizan estimaciones lo suficientemente precisas como para utilizarlos en cualquier herramienta de monitorización que precise el cálculo de la entropía de la distribución del tráfico de una red. En concreto ambos algoritmos son capaces de estimar la entropía de diferentes características con errores relativos por debajo del 5% en la mayoría de configuraciones evaluadas.

Sin embargo, el uso de memoria necesario para estimar la entropía muestra que el algoritmo de Sieving es mucho más eficiente que el algoritmo Streaming debido a que utiliza muchos menos contadores. Esto se debe fundamentalmente al hecho de que el número de contadores utilizados en el algoritmo de Streaming es logarítmicamente proporcional al tamaño del fragmento de tráfico de red del que se realiza la estimación, es decir, el número de contadores necesarios aumenta de forma más lenta a medida que se estima la entropía de un mayor número de paquetes manteniendo la precisión de la estimación. Esto provoca que para estimaciones de entropía de fragmentos de 100 milisegundos, el uso del algoritmo de Streaming sea muy ineficiente llegando incluso a necesitar más contadores que el cálculo exacto de la entropía, por lo que se puede concluir que el uso del algoritmo Sieving es el adecuado en el contexto de la aplicación CoMo.

Por otra parte, no se recomienda el uso de ninguno de los dos algoritmos para la estimación de la entropía de la distribución de los protocolos debido a que suelen utilizar muy pocos contadores para su cálculo. En particular, el cálculo exacto de la entropía y el algoritmo de Sieving han obtenido resultados muy parecidos ya que el comportamiento de Sieving cuando hay poca variación es similar al algoritmo que realiza el cálculo exacto. A modo de ejemplo, si en el contenido de la traza solo circula tráfico TCP y UDP se crearán únicamente dos contadores para ambos algoritmos. Sin embargo, el uso del algoritmo de Streaming para esta feature queda totalmente desaconsejado debido a que crea un contador por cada posición muestreada independientemente de la distribución del tráfico subyacente, lo que crea muchos más contadores de los realmente necesarios.

Por último, el coste computacional muestra los mismos resultados que el consumo de memoria. En la fase de tiempo real se observa que el algoritmo de Sieving va el doble de rápido aproximadamente, mientras que el comportamiento de Streaming es muy ineficiente debido a que se han utilizado fragmentos de tráfico pequeños. Por otra parte, la fase de post-procesado depende directamente del número de contadores de modo que el algoritmo de Sieving vuelve a demostrar ser más eficiente que el de Streaming utilizando aproximadamente el 20% de ciclos de CPU respecto al cálculo exacto de la entropía.

Como conclusión final, el uso del algoritmo de Sieving se adecua perfectamente al contexto de la aplicación CoMo, por lo que será el utilizado para evaluar el uso de la entropía como métrica para el sistema de gestión de recursos de esta herramienta.

Capítulo 4

Generador de anomalías

4.1 Introducción

En los últimos años se han investigado diversos enfoques para la detección automática de anomalías en el tráfico de red, tales como ataques de denegación de servicio, cortes de enlaces o escaneados de red. De hecho, se han desarrollado múltiples sistemas de detección de anomalías cuya efectividad es difícil de comprobar hoy en día. Una práctica común es evaluar estos sistemas mediante trazas de tráfico de redes públicas disponibles en Internet o a partir de trazas de ámbito privado capturadas en redes pertenecientes a universidades o a pequeños ISP. Sin embargo, ambas fuentes tienen sus desventajas: el proceso de anonimización y muestreo modifican las características del tráfico y por lo tanto los resultados; y el número de anomalías presentes en una traza de tráfico se encuentra limitado al contexto en el que se ha capturado por lo que es difícil encontrar trazas con múltiples anomalías o que estén libres de ellas.

Una alternativa al uso de trazas reales de red es el de simular dicho tráfico lo que permite tener un control total sobre las características de éste. Sin embargo, la simulación de patrones realistas de tráfico es un problema cuya solución hoy en día permanece sin ser resuelta por completo ya que se trata una tarea muy compleja. Es por ello por lo que se ha decidido aprovechar tanto el hecho de disponer de tráfico real como el de simular patrones de tráfico, realizando un herramienta que aúne las ventajas de ambos procesos para crear trazas lo más realistas posible con presencia de anomalías.

El generador de anomalías que se ha desarrollado es una herramienta capaz de inyectar anomalías definidas por el usuario en trazas de tráfico previamente capturadas. De esta forma, se hereda el control sobre el tráfico generado del ámbito de la simulación, mientras se mantiene el realismo proporcionado por la captura de tráfico. A pesar de las bondades de este acercamiento se sigue manteniendo la incertidumbre sobre la identificación de anomalías

presentes en el tráfico capturado, así como el hecho de evitar introducir errores en el diseño de anomalías.

La Figura 4-1 muestra la estructura del proceso de evaluación de una herramienta de administración de red con el uso de un generador de anomalías

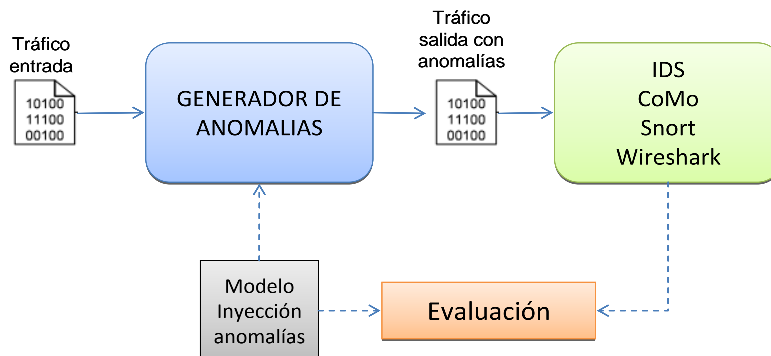


Figura 4-1 Proceso de evaluación de una herramienta de gestión de redes

El desarrollo de este generador de anomalías se ha basado en Flame [5] la cual es una herramienta desarrollada en C++ que permite la inserción de anomalías en ficheros Netflow [6]. El modelado de anomalías se realiza mediante scripts en Python [7] lo que facilita el desarrollo de nuevas anomalías y simplifica la distribución de la herramienta al no tener que compilar la aplicación cada vez que se desarrolla un nuevo modelo de anomalía.

Finalmente, se ha decidido escoger la librería Libpcap [8] como formato de fichero porque se trata de un estándar ampliamente utilizado y soportado por multitud de herramientas del ámbito de la administración de redes, tales como Wireshark [9], CoMo [10], Snort [11], entre otras.

4.2 Anomalías en redes

En el día a día se producen múltiples tipos de anomalías en Internet las cuales pueden ser debidas tanto a atacantes maliciosos que pretendan inhabilitar un servicio de red, como a errores en los propios dispositivos que forman Internet, como por ejemplo la avería de un nodo.

A continuación se presenta una clasificación de las anomalías que se pueden encontrar en una traza tráfico de red basada en el trabajo realizado en [12], junto con las características que se ven afectadas por este tráfico anómalo:

- Flujos Alfa: Este tipo de anomalía se ve reflejada en grandes volúmenes de flujo punto a punto. Se caracteriza por una mayor frecuencia de determinadas direcciones IP origen y destino.

- DoS: Denegación de servicio, ya sea desde un único origen o desde múltiples orígenes (DDoS). Su propósito es el de inhabilitar el acceso a un servicio mediante la sobrecarga del servidor atacado. Se caracteriza por predominar una dirección y puerto destino, mientras que el origen suele ser distribuido.
- Flash Crowd: Picos inusuales de tráfico a una sola dirección desde una típica distribución de orígenes. Suele deberse al uso por parte de muchos usuarios de un servidor determinado. Sus características son muy similares a un ataque DDoS
- Escaneo de puertos: Sondeo de múltiples puertos destino dentro de un conjunto pequeño de direcciones. El propósito de este ataque suele ser el de detectar si un conjunto pequeño de máquinas están ofreciendo determinados servicios asociados a puertos conocidos (RTP, HTTP,…). Suelen caracterizarse por una gran variedad de puertos destinos diferentes desde un origen fijo.
- Escaneo de redes: Sondeo de múltiples direcciones destino contra un pequeño conjunto de puertos. Similar al caso anterior, pero en este caso se pretende determinar qué máquinas están ofreciendo un determinado servicio.
- Outage Events: El tráfico es redireccionado debido a un error en algún equipo o por mantenimiento de la red. Suele ocasionar la perdida de numerosos paquetes por lo que el tiempo entre paquetes puede verse aumentado, o puede cambiar el número de saltos que deben realizar los paquetes.
- Punto a multipunto: Tráfico desde un origen a múltiples destinaciones, por ejemplo distribución de contenidos que implica grandes ráfagas desde una dirección origen a múltiples destinos todos al mismo puerto de alguna aplicación conocida.
- Gusanos: Se trata de un caso especial de escaneo de red en el cual se envían siempre el mismo contenido de datos ya que se replica código malicioso entre huéspedes.

4.3 Captura de tráfico de red

En la actualidad existen diferentes soluciones, tanto software como hardware, para la captura de tráfico de red. Es habitual que cada fabricante desarrolle sus propios formatos para la captura de tráfico, por lo que han acabado apareciendo múltiples tipos de fichero para almacenar esta información, por ejemplo los formatos ERF de Endance Measurament Systems o Netflow de Cisco Systems. En el mundo Open Source, un formato ha destacado por delante del resto: Libpcap.

A continuación se presenta una breve descripción de las principales características de estos tres formatos de fichero que son los más utilizados actualmente.

4.3.1 Libpcap

El formato de archivo libpcap es el principal formato utilizado por la mayoría de herramientas de red tales como Tcpdump, Wireshark o Snort. Se trata de un formato de fichero básico de captura de datos de red que se ha convertido en el estándar de facto de la captura de tráfico en UNIX y en el común denominador en el mundo Open Source.

Aunque normalmente se asume que este formato solo es útil para redes Ethernet, también puede servir para múltiples tipos de redes tales como ATM, Bluetooth, Frame Relay y VLAN, entre otras.

4.3.1.1 Formato de fichero

La extensión que habitualmente se utiliza para el almacenamiento de tráfico en formato Libpcap es *.pcap*. Al inicio de este fichero se encuentra una cabecera global con información general sobre el tráfico que se ha almacenado, seguida de un registro por cada paquete capturado. A su vez, cada registro se compone de una cabecera y el paquete de datos capturados. La estructura de un fichero PCAP es similar a la que se muestra en la Figura 4-2.

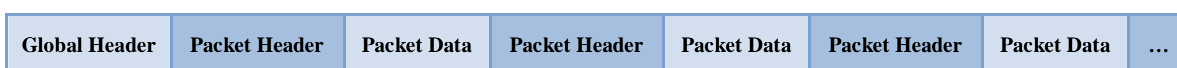


Figura 4-2 Contenido de un fichero en formato Libpcap

Respecto a los paquetes de datos cabe decir que no contienen necesariamente toda la información tal y como se transmite por la red; el paquete puede contener como máximo los primeros N bytes de cada paquete.

4.3.1.2 Cabecera global

Al inicio de todo fichero PCAP se encuentra la cabecera global del fichero pcap. La Figura 4-3 muestra el contenido de esta cabecera:

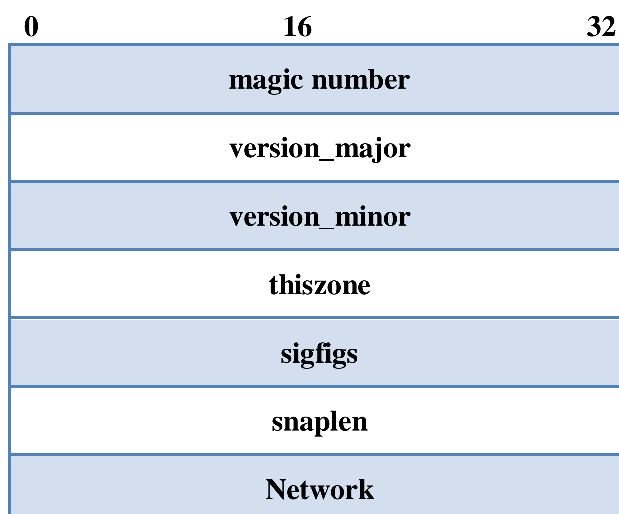


Figura 4-3 Cabecera global de Libpcap

- **magic number:** se utiliza para detectar el formato del fichero y el orden de los bytes de la cabecera y su valor es 0xa1b2c3d4 en el orden nativo de máquina. La aplicación que lea el fichero PCAP, comprobará el orden de este número para saber si debe intercambiar el orden de los bits del resto de campos de la cabecera.
- **version_major, version_minor:** número de versión del fichero.
- **thiszone:** corrección temporal entre GMT (UTC) y la zona temporal de los siguientes timestamps. En la práctica siempre es porque los instantes de tiempo están GMT (UTC).
- **sigfigs:** en teoría este campo contiene la precisión de los timestamps, pero en la práctica todas las herramientas ponen este campo a 0.
- **snaplen:** tamaño máximo de datos capturados por paquete. Habitualmente su valor es superior 65535.
- **network:** tipo de datos de nivel de enlace, por ejemplo, este campo vale 1 cuando se trata de un paquete en una red Ethernet.

4.3.1.3 Cabecera de Registro o de paquete

Cada paquete capturado viene precedido de una cabecera con 4 campos. La Figura 4-4 muestra su contenido:

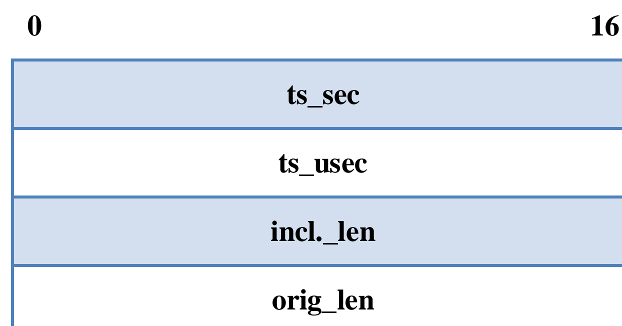


Figura 4-4 Cabecera de paquete Libpcap

- **ts_sec:** fecha y hora de captura del paquete en segundo desde el 1 de Enero de 1970 a las 00:00:00 GMT.
- **ts_usec:** microsegundos del instante de tiempo en que el paquete ha sido capturado como un offset de ts_sec. Este valor no debe sobrepasar 1.000.000 ya que en ese caso ts_sec debe ser incrementado.
- **incl_len:** número de bytes capturados del paquete y guardados en el fichero. Este valor nunca debe ser superior a orig_len o snaplen.
- **orig_len:** tamaño del paquete tal y como aparece en la red.

4.3.1.4 Paquete de datos

El paquete capturado se guarda inmediatamente después de la cabecera y se escriben en el fichero tantos bytes como se indica en *incl_len* bytes sin ningún tipo de alineación u ordenación. Por otra parte, el número máximo de bytes no puede sobrepasar el valor determinado en el campo *snapshot length* que viene definido en la cabecera global del paquete *.pcap*. Habitualmente se utiliza un valor de 65535 para asegurarnos de que ningún paquete será recortado durante la captura de datos.

4.3.2 Netflow

NetFlow es un protocolo de red desarrollado por Cisco Systems que funciona sobre equipos con sistema operativo Cisco-IOS que se utiliza para la recogida de información de tráfico IP. Es de código propietario pero recibe el apoyo de otras plataformas, tales como routers Juniper, Linux o FreeBSD y OpenBSD.

A pesar de que inicialmente fue implementado por Cisco, Netflow está emergiendo como un estándar del grupo de estandarización IETF[1]: Internet Protocol Flow Information eXport (IPFIX). Este estándar está basado en la implementación de la versión 9 de Netflow y ya está siendo añadido a los nuevos dispositivos desarrollados por los principales vendedores de equipamiento de red.

4.3.2.1 Flujo de red

Un flujo de red se ha definido de muchas maneras. La definición tradicional de Cisco es utilizar una 7-tupla, donde el flujo se define como una secuencia unidireccional de todos los paquetes que comparten los siguientes 7 valores:

1. Dirección IP de origen.
2. Dirección IP destino.
3. Puerto de origen para UDP o TCP, 0 para otros protocolos.
4. Puerto de destino para UDP o TCP, 0 para otros protocolos.
5. Protocolo IP
6. Interfaz de entrada (SNMP ifIndex)
7. Tipo de servicio IP

[1] IETF Internet Engineering Task Force

Los routers Cisco que tienen habilitada la generación de registros Netflow envían paquetes mediante el uso de UDP o SCTP que son recogidos por los colectores de Netflow. La Figura 4-5 se muestra el esquema de funcionamiento de recogida de información en Netflow.

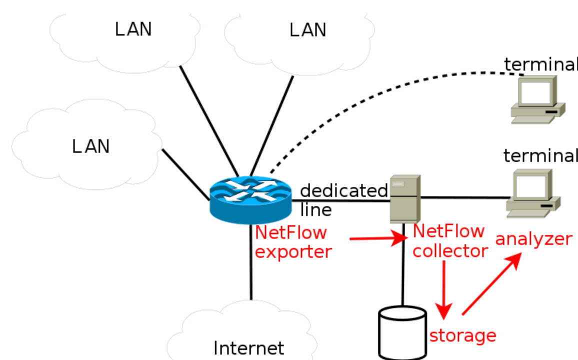


Figura 4-5 Esquema de funcionamiento de recogida de información en Netflow

El router genera registros Netflow cuando determina la finalización del flujo que cumple con la 7-tupla. Esto lo hace mediante el uso de un contador de envejecimiento: cuando el router detecta nuevo tráfico para un flujo ya existente, el router resetea este contador. Además, la detección de finalización de una sesión TCP también causa que el router de por finalizado el flujo. Estos routers también pueden ser configurados para que generen un registro de flujo cada cierto intervalo de tiempo incluso si el flujo todavía no ha finalizado. Una variante de Netflow conocida como Flexible Netflow (FNF) permite además definir los parámetros que configuran cada flujo de datos.

4.3.2.2 Registros Netflow

El contenido de un registro Netflow puede contener una gran variedad de información del tráfico de un flujo determinado. Existen diferentes versiones de Netflow pero las más utilizadas comúnmente son la versión 5 y la versión 9.

El contenido de un registro Netflow versión 5 es el siguiente:

- Número de versión.
- Número de secuencia.
- Índices de los interfaces de entrada y salida utilizados por SNMP.
- Timestamps de los tiempos de inicio y fin del flujo en milisegundos desde el último arranque del dispositivo.
- Número de bytes y paquetes observados en el flijo.
- Cabeceras de nivel 3:
    - Direcciones IP origen y destino.
    - Puertos origen y destino.
    - Protocolo IP

  - Tipo de servicio (ToS)
- En caso de ser un segmento TCP, la unión de todos los flags TCP observados en todo el flujo.
- Información de enrutamiento de nivel 3.
  - Dirección IP del siguiente salto a lo largo de la ruta hacia el destino.
  - Máscara IP origen y destino (en notación CIDR).

La versión 9 de este formato incluye estos mismos campos, pero a diferencia de la versión 5, también puede incluir información relativa a otros protocolos tales como MLPS o IP versión 6.

4.3.3 DAG Cards Extensible Record Format – ERF

Las tarjetas de monitorización DAG de Endace Measurement Systems producen ficheros en su propio formato nativo, conocido como Extensible Record Format o ERF [13]. Lo ficheros ERF contienen una serie de registros ERF con cada uno de ellos describiendo un paquete de datos.

Un fichero ERF consiste solo en registros ERF, evitando cualquier tipo de cabecera especial y permitiendo a su vez realizar de forma arbitraria concatenaciones o divisiones en los límites de cualquier registro ERF de un fichero.

ERF soporta múltiples tipos de conexiones asignando un valor para cada una de ellas. Un ejemplo de los tipos que soporta ERF son: IPv4, Ipv6, ATM, HDLC, AAL2 entre muchos otros.

4.3.3.1 Cabecera genérica

Cada registro ERF consta de una cabecera junto con el flujo de datos capturado que consta de una serie de campos comunes a todos los registros. El ordenamiento de bytes de estos campos es el mismo que el del tráfico de red, big-endian, exceptuando los campos con timestamps que se encuentran en little-endian. Los datos son capturados como un flujo de bytes sin ningún tipo de reordenación, es decir, en el el orden de red.

Un ejemplo de cabecera se puede ver en la Figura 4-6:

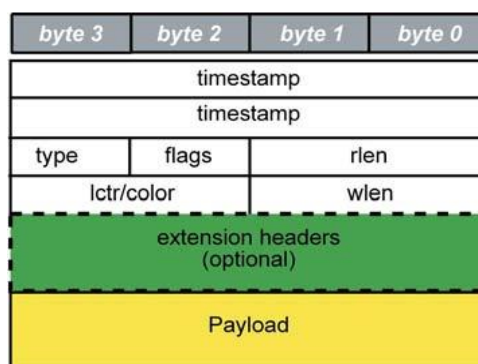


Figura 4-6 Cabecera ERF

- **Timestamp:** Instante de tiempo de la llegada del paquete en 64 bits.
- **Type:** Tipo de extensión de cabecera.
- **Flags:** Conjunto de 8 bits que indican diferentes propiedades del registro tales como el número de interfaz de captura de la tarjeta DGAG, tamaño variable del registro ERF, si el registro ha sido truncado o si ha habido algún error.
- **rlen:** Tamaño del registro en bytes.
- **lctr/color**
- **wlen:** Tamaño de los capturados en bytes.
- **Extension headers:** Cabecera extra que permite introducir información adicional relativa al paquete capturado. Puede haber más de una cabecera extra.
- **Payload:** Datos capturados en el el registro.

4.3.3.2 ERF Timestamps

ERF incorpora un timestamp generado por hardware con el instante de tiempo de la llegada del paquete. Este timestamp consiste un número de punto fijo de 64 bits en little-endian que representa la parte entera y la parte fraccionaria del número de segundos transcurridos desde la medianoche del 1 de enero de 1970.

Los primeros 32 bits contienen el número entero de segundos mientras que los siguientes 32 bits contienen la fracción de segundos. Esto permite una resolución máxima de 2^{-32}, o lo que es lo mismo, una resolución de 233 picosegundos.

Otra ventaja de este formato de timestamp es que la diferencia entre dos instantes de tiempo se puede calcular con una simple resta de dos valores de 64 bits, a diferencia de las estructuras de tiempo de UNIX en las cuales se ha de tener en cuenta posibles “overflows”.

4.4 Flame

Para el desarrollo de nuestro generador de anomalías nos hemos basado en Flame el cual es una herramienta de modelado de anomalías en formato Netflow desarrollado por miembros del Instituto Federal Suizo de tecnología de Zurich y que facilita la evaluación de sistemas de detección de anomalías.

El núcleo de Flame esta desarrollado en C++ por cuestiones de rendimiento. Además, Flame utiliza el lenguaje de scripting Python (embedded python) para la parte de modelado de anomalías. Esto permite desarrollar nuevas anomalías sin necesidad de recompilar toda la aplicación.

La aplicación está formada por una serie de módulos, que se comunican entre si mediante el uso de pipes con nombre. Los módulos disponibles en Flame son: Netflowv5/v9Reader, Netflowv5/v9Writer, FlowGenerator, FlowDeleter y FlowMerger.

La Figura 4-7 muestra un ejemplo de la estructura que sigue esta herramienta para generar una serie de anomalías.

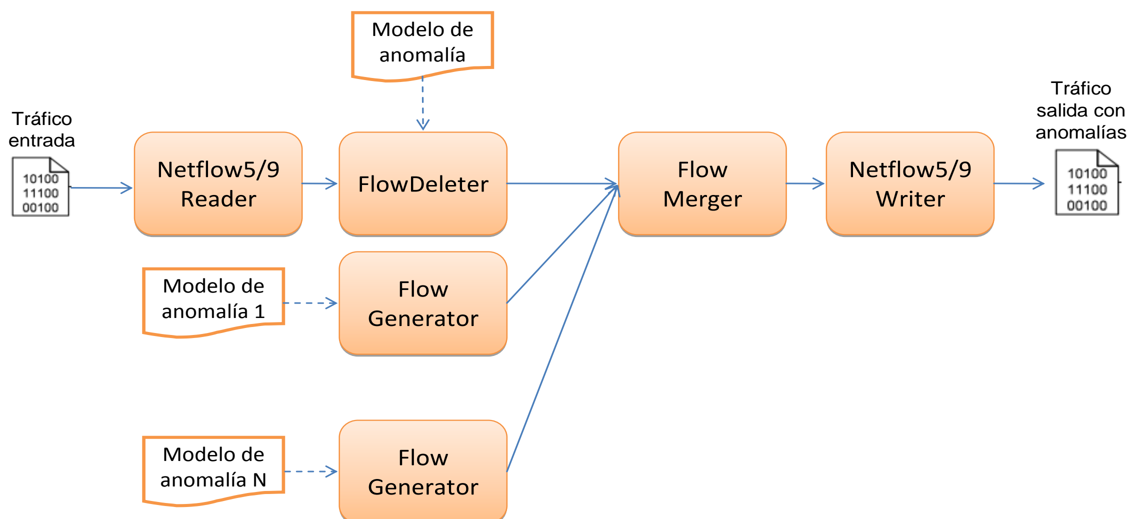


Figura 4-7 Estructura de generación de una anomalía con FLAME

La facilidad en la implementación de anomalías es uno de los principales objetivos de nuestro generador. Por este motivo se ha utilizado como base FLAME ya que cubre con creces este propósito debido al uso de scripts muy simples en Python para modelar cualquier tipo de anomalías.

4.5 Desarrollo del generador de anomalías

4.5.1 Descripción

Nuestro generador de anomalías sigue el mismo patrón que el visto en la herramienta FLAME donde se aprovecha la estructura de los diferentes procesos y el modelado de anomalías a través de scripts en Python. La Figura 4-8 muestra un ejemplo de una posible configuración del generador de anomalías y la relación ente los diferentes ficheros, procesos y scripts.

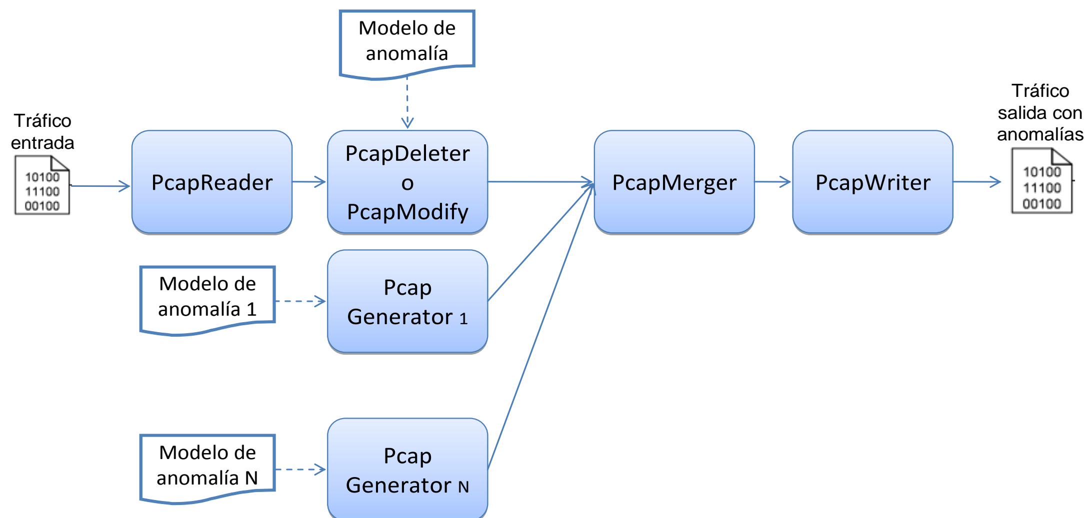


Figura 4-8 Estructura de la herramienta de generación de anomalías

Al igual que FLAME, nuestro generador consiste en una serie de módulos desarrollados en C++ los cuales se comunican entre ellos mediante el uso de pipes con nombre y una serie de scripts en Python que modelan las diferentes anomalías.

En la figura anterior se puede observar que el generador permite introducir más de una anomalía en la misma ejecución lo que permite una mayor facilidad y flexibilidad en el momento de crear múltiples anomalías sobre un mismo fichero PCAP.

A continuación se describen las funciones que realizan los diferentes módulos que componen el generador de anomalías junto con los parámetros que se le deben pasar a cada uno:

4.5.1.1 PcapReader

PcapReader *FICHERO_ORIGEN PIPE_DESTINO*

Ejecutable que se encarga de leer el contenido de FICHERO_ORIGEN en Libpcap paquete por paquete e ir escribiendo cada uno de ellos en la PIPE_DESTINO. Para maximizar la compatibilidad de formatos se han utilizado las funciones de la librería Libpcap de UNIX para la lectura del fichero origen.

Antes de enviar el paquete a la pipe se escribe un registro con información sobre el contenido del paquete. La información que se escribe es la siguiente:

- Dirección IP origen
- Dirección IP destino
- Puerto origen
- Puerto destino
- Protocolo

- Número de paquetes
- Número de bytes
- Número de segundos del instante de tiempo
- Número de microsegundos del instante de tiempo
- TCP flags
- Tipo de paquete

4.5.1.2 PcapWriter

PcapWriter *PIPE_ORIGEN FICHERO_DESTINO*

Ejecutable que lee el contenido de la PIPE_ORIGEN y lo escribe en el FICHERO_DESTINO mediante la librería Libpcap de UNIX. En la pipe origen ya se encuentran todos los paquetes ordenados. El ejecutable terminará cuando detecte que ya no queda nada por leer de la pipe.

4.5.1.3 PcapMerger

PcapMerger *PIPE_DESTINO PIPES_ORIGEN*

Ejecutable que escribe en la PIPE_DESTINO el contenido de las pipes indicadas en el resto de parámetros. Este proceso se encarga de ordenar los paquetes según el instante de tiempo indicado en el registro interno que precede a cada paquete. La resolución de estos timestamps es de 1 microsegundo.

Es importante destacar que la cabecera global de Libpcap debe estar en la primera pipe de lectura –segundo parámetro- ya que solo la buscará en este pipe. Si no la encontrase, el generador de anomalías no la creará y por tanto, el fichero final no será legible por otras aplicaciones.

4.5.1.4 PcapGenerator

PcapGenerator *PIPE_DESTINO SCRIPT*

PcapGenerator es el encargado de generar anomalías en función del modelo que se indica en SCRIPT pasado por parámetro. Estos paquetes se almacenan en una cola con prioridad así que es necesario que el timestamp se vaya incrementando con cada paquete. A medida que se van generando paquetes, estos se escriben en la PIPE_DESTINO precedidos por el registro interno.

4.5.1.5 PcapDeleter

PcapDeleter *PIPE_ORIGEN PIPE_DESTINO SCRIPT*

Este ejecutable se encarga de leer la información de los registros que preceden a cada paquete y decidir en función de lo indicado en SCRIPT si debe eliminar el paquete siguiente o no. Si no se elimina el paquete se escribe en la PIPE_DESTINO exactamente igual que como se leyó.

4.5.1.6 PcapModifier

PcapModifier *PIPE_ORIGEN PIPE_DESTINO SCRIPT*

PcapModifier lee registros de la PIPE_ORIGEN y decide si debe modificarlos o no en función de lo indicado en el SCRIPT. El paquete se modifica según lo indicado en el script y se pasa a la PIPE_DESTINO con los cambios realizados.

4.5.2 Modelado de anomalías

Uno de los requisitos del generador es que el desarrollo y modelado de anomalías sea rápido, sencillo y flexible. Como ya se ha indicado, la herramienta FLAME resolvía esto mediante el uso de scripts en Python los cuales son leídos por los diferentes procesos e interpretados mediante el uso de las librerías Boost [14] . En el generador de anomalías desarrollado se sigue esta misma política con la diferencia de que se ha adaptado al contexto de generación de paquetes en lugar de flujos Netflow.

Existen tres clases diferentes en Python para cada uno de los ejecutables que requieren como parámetro un script – PcapGenerator, PcapDeleter y PcapModifier –. A su vez, cada clase puede llamar a una serie de funciones que están implementadas en cada uno de los módulos asociados con las que se puede realizar acciones tales como generar, eliminar o modificar paquetes.

A continuación se realiza una descripción de cada una de las clases que se deben implementar en los diferentes scripts en Phyton junto con las funciones que deben ser implementadas y las que pueden ser llamadas.

4.5.2.1 PythonPcapGenerator

Esta clase está asociada al módulo PcapGenerator y es la encargada de establecer el patrón de generación de paquetes junto con sus propiedades. Para que el generador funcione correctamente el usuario debe desarrollar dentro de esta clase las funciones *getRequest* y *getReply* que pueden realizar cualquier tipo de acción en función de la anomalía que se desee crear.

Por otra parte, este script dispone de las funciones *sendTCP* y *sendUDP* que crean un paquete cada vez que son llamadas con las propiedades especificadas. La generación de la anomalía finalizará cuando el usuario haga la llamada a la función *stop*.

4.5.2.2 PythonPcapDeleter

Esta clase está asociada al módulo PcapDeleter y es la encargada eliminar paquetes de la traza de entrada en función del patrón que se defina en el script. En esta clase se debe implementar la función *delete* ya que es la que utilizará el módulo PcapDeleter para establecer si un paquete debe ser eliminado o no. Una vez se ha determinado el futuro del paquete el usuario deberá

llamar a la función *delRec* que indicará al módulo PcapDeleter que el paquete que se está leyendo debe ser eliminado – no se copiará a la *pipe* de salida –.

A parte de la función *delRec*, esta clase dispone de una serie de funciones que le permiten conocer las propiedades del paquete actual. En definitiva, las funciones disponibles en esta clase son:

- **delRec:** Indica que se debe eliminar el paquete.
- **get_srcAddr:** Proporciona la dirección IP origen.
- **get_dstAddr:** Proporciona la dirección IP destino.
- **get_sec:** Proporciona el instante de tiempo en segundos
- **get_usec:** Proporciona el instante de tiempo en microsegundos.
- **get_srcPort:** Proporciona el puerto origen.
- **get_dstPort:** Porporciona el puerto destino.
- **get_tcpFlags:** Proporciona los flags del protocolo TCP.
- **get_prot:** Proporciona el protocolo que transmite el paquete IP.
- **get_length:** Proporciona el tamaño del paquete.

A diferencia de PcapGenerator, en esta clase no se determina el final de la anomalía ya que el módulo PcapDeleter está configurado para que lea toda la información de su entrada y la escriba a la salida a excepción de los paquetes eliminados.

4.5.2.3 PythonPcapModifier

El funcionamiento de esta clase es homólogo al de la anterior. En este caso, se debe implementar la función *modify* la cual determinará si un paquete se debe modificar. La principal diferencia radica en que en este caso se dispone de la función *modRec* que permite modificar el paquete que se está leyendo y requiere que se le pasen como parámetro las principales propiedades del paquete – dirección IP origen y destino, puerto origen y destino, tamaño del paquete, instante de tiempo, flags tcp y el protocolo. Al igual que la clase anterior dispone de una serie de funciones que proporcionan información sobre el paquete actual que se está leyendo para poder decidir si se debe modificar o no – get_srcAddr, get_dstAddr, etc. –.

4.5.3 Generación de una anomalía

La ejecución de la herramienta se realiza mediante un Shell Script de UNIX siguiendo la estructura presentada en la Figura 4-8. En este script se deben especificar tanto el fichero de entrada de datos como el de salida, crear las *pipes* necesarias para la interconexión de los diferentes módulos así como los scripts que utilizarán cada uno de ellos.

A continuación se muestra un script a modo de ejemplo donde se introducen 3 anomalías diferentes: un ataque por denegación de servicio, un escaneo de puertos y el corte de un enlace con una dirección origen determinada.

```
#!/bin/sh

# Injects two DDoS attacks and modify defined source IP address
# into a given libpcap trace

INPUT="input.pcap"
OUTPUT="ouput.pcap"
SCRIPT1="delete_addr.py"
SCRIPT2="DDoS.py"
SCRIPT3="port_scan.py"

# Create pipes
mkfifo pipe1
mkfifo pipe2
mkfifo pipe3
mkfifo pipe4
mkfifo pipe5

# Execute applications
PcapReader $INPUT pipe1  &
PcapModifier pipe1 pipe2 $SCRIPT1 &
PcapGenerator pipe3 $SCRIPT2 &
PcapGenerator pipe4 $SCRIPT3 &
PcapMerger pipe5 pipe2 pipe3 pipe2 &
PcapWriter pipe5 $OUTPUT

# Delete pipes
rm pipe1
rm pipe2
rm pipe3
rm pipe4
rm pipe5
```

En este script se puede observar como se crean y se destruyen cada una de las pipes y como se pasan los diferentes parámetros para que la herramienta sea capaz de generar las anomalías especificadas en cada uno de los scripts, para finalmente almacenar la nueva traza en el fichero de salida.

Como se ha visto a lo largo de este punto y a modo de resumen, se puede decir que la estructura de este generador de anomalías se divide en tres tipos de componentes: los scripts en Python que son los encargados de establecer como se van a generar las anomalías – modelo y propiedades –; los ejecutables que se encargarán de realizar las acciones que los scripts indican – crear, destruir o modificar paquetes –; y el Shell Script con las instrucciones de que anomalías se van a generar junto con el origen y el destino de los datos.

4.6 Conclusiones

Uno de los principales objetivos de este Proyecto de Final de Carrera era el de crear una herramienta capaz de modificar trazas de tráfico en Libpcap ya existentes y añadirles anomalías modeladas por el usuario. Mediante el generador de anomalías desarrollado se ha conseguido cumplir este objetivo a partir de la herramienta FLAME y manteniendo una gran sencillez y flexibilidad en la creación de nuevas modelos de anomalías.

Además se ha desarrollado la herramienta haciendo uso de las librerías de programación de Libpcap de forma que se pueda mantener la compatibilidad de la lectura y escritura de trazas de red con futuras versiones de este estándar.

Por otra parte, se podría haber mantenido la compatibilidad con ficheros Netflow pero esto hacía que la herramienta fuese más lenta -se modifican trazas de tráfico con millones de paquetes-, por lo que se ha decidido separar ambos formatos para poder maximizar la velocidad del generador de anomalías.

Finalmente, es importante destacar que la herramienta está siendo utilizada con éxito por los estudiantes de postgrado del CCABA del Departamento de Arquitectura de Computadores de la UPC.

Capítulo 5

Load Shedding y uso de la entropía

5.1 Introducción

La monitorización y el análisis de tráfico de red en tiempo real son mecanismos cruciales para la gestión y administración de redes de datos que se encuentran en funcionamiento, motivo por el que los operadores de red necesitan obtener métricas que les aporten información sobre el tráfico que la atraviesa. Para satisfacer esta demanda de información se requieren sistemas capaces de hacer frente a los efectos de situaciones con sobrecarga debido a, por ejemplo, largos volúmenes de datos, a altas velocidades de transmisión, a la propia naturaleza a ráfagas del tráfico de red y al aumento de la velocidad de los nuevos enlaces.

La herramienta CoMo implementa un mecanismo que trata de sobreponerse a estas situaciones de cargas extremas de tráfico con el objetivo de mantener unos niveles aceptables de precisión en las métricas que se le solicitan. Esta técnica se denomina Load Shedding y una de las principales novedades que introduce es la capacidad de funcionar correctamente sin ningún conocimiento previo de las peticiones que va a recibir. En lugar de esto, el módulo extrae un conjunto de características del flujo de tráfico, también llamadas *features*, para construir un modelo de predicción en tiempo real del consumo de recursos de las peticiones que realizan los usuarios. Con este modelo el sistema es capaz de decidir si debe muestrear la entrada de paquetes y que cantidad debe muestrear.

Actualmente el módulo de gestión de recursos se basa en el cálculo de 55 features básicas tales como el número de paquetes, el número de bytes, el número de direcciones IP diferentes o el número de direcciones IP únicas, entre otras. A partir de la relación de estas métricas y el consumo pasado de CPU de los diferentes módulos, el algoritmo de Load Shedding es capaz de predecir la carga de recursos que necesitarán el sistema y así poder tomar la decisión de si se debe aplicar o no un muestreo a la entrada datos.

En este capítulo se pretende observar si utilizando la entropía como nueva feature podemos obtener mejores estimaciones del uso de recursos del sistema Load Shedding, tanto en ausencia como en presencia de anomalías. Para ello se ha utilizado el segundo algoritmo de estimación de la entropía explicado en el capítulo 3, el algoritmo de Sieving, ya que se ha demostrado que obtiene buenos resultados en la estimación de la entropía manteniendo un error relativo bajo con fragmentos pequeños de tráfico como los de 100 milisegundos que utiliza CoMo.

Además, para la inserción de las anomalías se ha utilizado el generador explicado en el capítulo 4, simulando 3 tipos de anomalías diferentes, un ataque de denegación de servicio, un ataque de escaneo de puertos y una anomalía donde la distribución de los flujos IP tiene valores máximos y mínimos de la entropía.

5.2 CoMo

CoMo (Continuos Monitoring) es un sistema de monitorización pasivo diseñado por los laboratorios de investigación de Intel en Cambridge, en colaboración con las universidades de Cambridge, Pisa y Politécnica de Cataluña. Se trata de una herramienta flexible, escalable y que no precisa de un equipo potente para funcionar. Una de las características de CoMo es que permite a los usuarios añadir funcionalidades mediante el desarrollo de módulos que calculan cualquier tipo de métricas del tráfico observado, ya sean en tiempo real, o a partir de trazas de red capturadas en diferentes formatos (ERF, Pcap, Netflow, etc). Su código se distribuye bajo licencia BSD.

5.2.1 Estructura del sistema

El sistema está compuesto por dos tipos de componentes principales: los procesos núcleo que controlan los datos dentro del sistema encargándose de la captura, exportación y almacenamiento de paquetes, así como la gestión de las consultas de los usuarios y de los recursos; y los plugins o módulos añadidos que son los responsables de ejecutar operaciones específicas sobre los datos de entrada en función de lo que desee el usuario.

La Figura 5-1 presente en el The CoMo White Paper [15] muestra el flujo de datos dentro de la herramienta CoMo. Las cajas blancas representan los plugins, mientras que las cajas grises corresponden a los procesos núcleo.

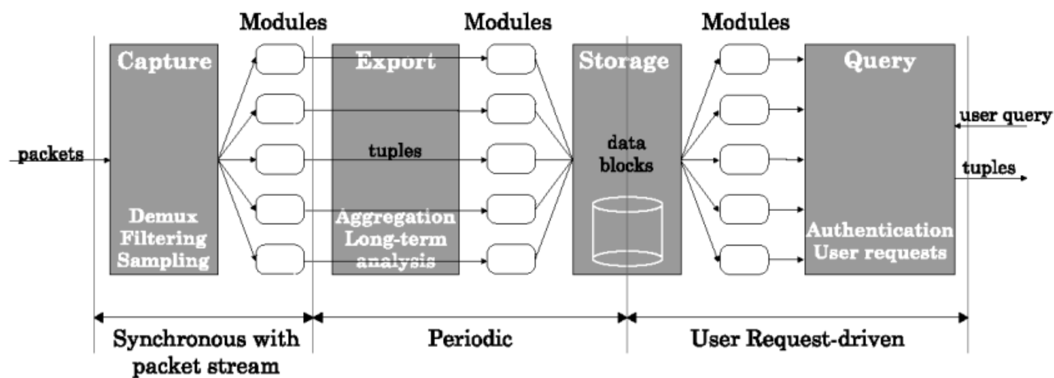


Figura 5-1 Estructura de la herramienta CoMo

El funcionamiento es el siguiente: Por una parte el sistema captura los paquetes del enlace monitorizado o del fichero de captura. Estos paquetes son procesados por los sucesivos procesos núcleos hasta que los datos se guardan en el disco. Por otra parte, los módulos reciben las peticiones de los usuarios y estos acceden al disco para recuperar la información necesaria.

Para determinar la configuración inicial de los procesos núcleo y de los módulos se lee el fichero de configuración *como.conf* donde se especifican los métodos de captura de paquetes, los parámetros de los diferentes procesos núcleos, así como los módulos y filtros que estarán activos y sus argumentos.

5.2.2 Procesos núcleo

Los procesos núcleo se encargan de las acciones necesarias para capturar los datos de entrada, ya sea de un enlace físico o de un fichero capturado previamente, y almacenarlos en el disco duro. La comunicación entre los diferentes procesos núcleo se realiza mediante un sistema de paso de mensajes que permite a la herramienta poder funcionar de forma distribuida mediante el uso de un conjunto de servidores, también llamado cluster.

Los cinco procesos núcleo principales que dispone CoMo son los siguientes:

- **Capture:** Es el proceso encargado de la captura y filtrado de los paquetes para posteriormente proporcionarlos a los diferentes módulos. Es capaz tanto de leer una tarjeta de captura de red como un fichero de captura de datos en múltiples formatos de los que se puede destacar Libpcap, ERF, Netflow y BPF entre otros.
- **Export:** Permite el análisis del tráfico a largo plazo y dar información adicional de la red, como por ejemplo, tablas de rutas.
- **Storage:** Planifica i gestiona los accesos al disco por parte de los diferentes módulos para evitar solapamientos.
- **Query:** Recibe las peticiones de los usuarios, aplica las transformaciones sobre el tráfico o sobre los resultados pre-calculados y devuelve los datos al usuario.

- **Supervisor:** Es el responsable de gestionar las excepciones que pueda aparecer, por ejemplo fallos en procesos, y decidir cuando cargar, parar o iniciar un módulo dependiendo de los recursos disponibles o de las políticas de acceso.

5.2.3 Módulos

Los módulos son desarrollados por el propio usuario de forma que es él mismo el que especifica que información se debe recolectar. Se pueden interpretar como una pareja filtro-función, donde el filtro especifica sobre que paquetes se aplicará la función. Por ejemplo, si la métrica de tráfico es *calcular el número de paquetes que tienen como destino el puerto 80*, el filtro se configura para capturar solamente los paquetes que tienen como destino el puerto 80, mientras que la función consistirá en incrementar un contador por cada paquete que se captura. Cabe destacar que no todos los módulos calculan necesariamente estadísticas de los paquetes de entrada, sino que también puedes transformar el tráfico de entrada, como por ejemplo, pasar un conjunto de paquetes a un flujo IP.

Otra característica de este diseño es que cada módulo es independiente del resto de módulos ya que no existe la posibilidad de que se comuniquen entre sí, por lo que no pueden compartir información. Este hecho simplifica la gestión de recursos de CoMo, pero por otro lado introduce una redundancia en el sistema ya que es posible que diferentes módulos realicen los mismos cálculos de los datos de entrada. A pesar de ello existe una método para que un módulo utilice la información obtenida por otro, que no es más que ejecutar dos instancias de CoMo utilizando los resultados de la primera como entrada de datos para la segunda. O dicho de otro modo, se utilizan los datos obtenidos por el módulo de la primera instancia como entrada para el módulo de la segunda instancia.

5.2.4 Interfaz gráfica: CoMo-Live!

La herramienta CoMo dispone de una interfaz gráfica basada en Web e implementada en PHP4. Esta interfaz permite ver de forma gráfica los resultados de las consultas que se están ejecutando en los diferentes módulos de CoMo

Para que funcione correctamente es necesario que los módulos implementen la función *callback* print de una forma diferente para que CoMo-Live! pueda obtener los datos para mostrar la gráfica. La Figura 5-2 muestra una captura de esta interfaz para un módulo que calcula la velocidad del enlace en Mbps y en número de paquetes por segundo:

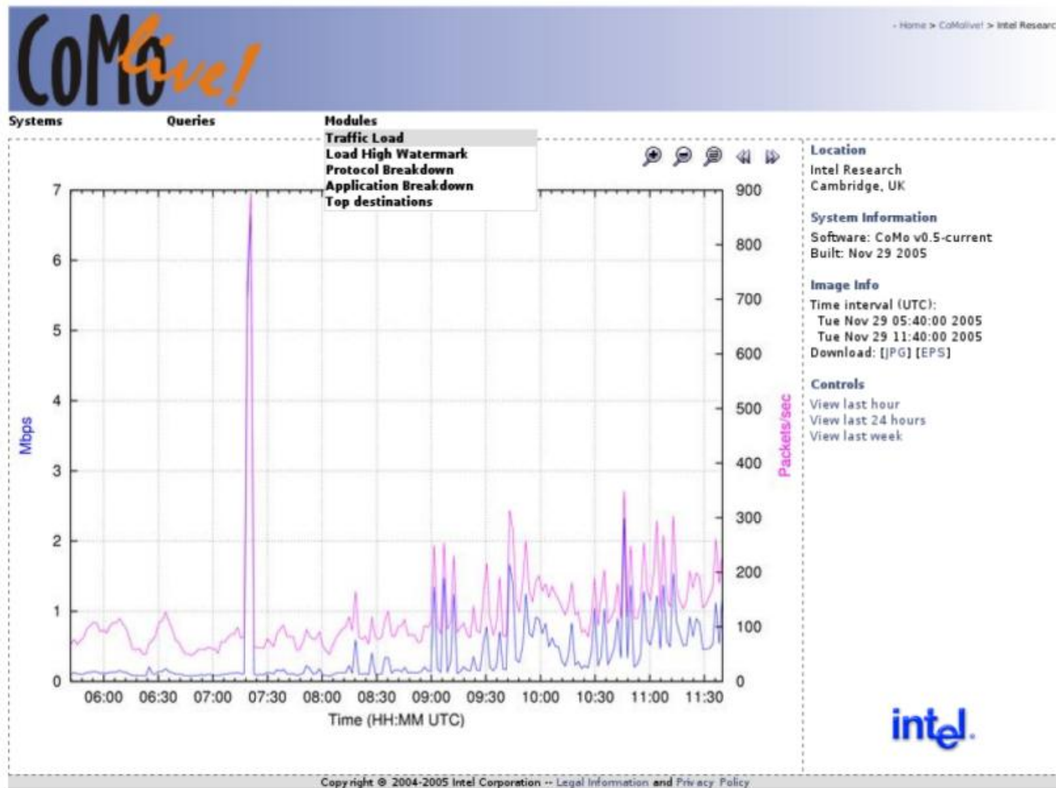


Figura 5-2 Interfaz gráfica CoMo-Live!

5.3 Gestión de recursos en CoMo

5.3.1 Introducción

Un sistema de monitorización de red que pretenda ser robusto necesita de algún sistema capaz de hacer frente a las sobrecargas de información que pueda sufrir, ya sea debido a trabajar en enlaces de alta capacidad o al incremento de la complejidad y de la precisión de las solicitudes que se realizan a estos sistemas. Por ejemplo, existe un aumento de la demanda de aplicaciones de monitorización que requieren el seguimiento e inspección de un gran número de conexiones de red concurrentes para la detección de intrusiones y anomalías. Esto ha llevado a un aumento de los requisitos de procesamiento de las herramientas de monitorización de redes y al desarrollo de diferentes mecanismos para evitar estos posibles problemas.

Una posible forma de abordar esta situación sería la de sobredimensionar el sistema donde se ejecuta la herramienta de monitorización de forma que pudiese soportar picos de tráfico o cualquier tipo de solicitud por compleja que sea. Sin embargo, esta solución sería extremadamente cara y con una infraestructura en general infrautilizada, por lo que múltiples investigaciones han tratado este reto desde otro enfoque.

La mayoría de las propuestas existentes están basadas en técnicas de reducción de datos, tales como el filtrado, la agregación o el muestreo de paquetes. El ejemplo más representativo de esto podría decirse que es el protocolo de Cisco Netflow el cual agrupa los paquetes de entrada en registros Netflow. Otros ejemplos son Adaptative Netflow el cual adapta el ritmo de muestreo al consumo de memoria o el sistema de monitorización desarrollado por Keys et al [16] que combina agregación, muestreo adaptativo y uso eficiente de algoritmos eficientes de cuenteo para extraer un conjunto predefinido de 12 características del tráfico.

Además, el diseño de mecanismos para soportar situaciones de sobrecarga es un problema clásico en cualquier sistema que trabaje con información en tiempo real y es por ello que se han propuesto soluciones para otros tipos de entornos. Por ejemplo, en la comunidad de bases de dato, el sistema Aurora [17] se desprende del exceso de carga insertando operadores de descarte en el flujo de petición de datos, mientras que el sistema TelegraphCQ [18] utiliza técnicas de procesamiento de peticiones para proporcionar respuesta con un retraso limitado en presencia de sobrecarga.

En general, este tipo de soluciones suelen presentar dos tipos de limitaciones: por un lado limitan el tipo de métricas que se pueden extraer del flujo de tráfico limitando los posibles usos de los sistemas; por otra parte, estas soluciones asumen algún tipo de conocimiento explícito sobre el coste y la selectividad de cada operador, requiriendo un coste de tiempo en la fase de diseño e implementación de cada uno de ellos.

La herramienta de monitorización CoMo utiliza el método denominado Load Shedding desarrollado por Pere Barlet, et al [19] que le permite soportar situaciones de sobrecarga evitando la perdida descontrolada de paquetes y mantenido una mínima precisión en el resultado de las peticiones que recibe la aplicación. Este sistema utiliza actualmente un conjunto de características del tráfico de red basadas en contadores muy simples para predecir el consumo de CPU, por lo que el objetivo de esta capítulo será el comprobar si mediante el uso de los algoritmos de estimación de la entropía descritos en el capítulo 2 se puede mejorar los sistemas de predicción del módulo de Load Shedding de CoMo.

5.3.2 Subsistema de predicción y control de carga

La principal novedad de este sistema es que es capaz de funcionar sin ningún conocimiento explicito de las peticiones que cursará el sistema ni del tipo de cálculos que realizará – por ejemplo, clasificación de flujos, búsqueda de cadenas de caracteres, etc –. De esta forma se mantiene la flexibilidad del sistema de monitorización, permitiendo el rápido desarrollo e implementación de nuevas aplicaciones de monitorización de red.

El funcionamiento del sistema se basa en estimar el uso de recursos que se va a realizar a partir de la relación de un predeterminado conjunto de características, o *features*, del tráfico de entrada y el uso de recursos actual. Una feature es un valor que describe una propiedad específica de una secuencia de paquetes, como por ejemplo el número de paquetes o el número único de direcciones IP origen entre otras. Una de las ventajas de estas features es que su cálculo es sencillo y con un coste computacional máximo determinado. La Figura 5-3 [19] muestra los cuatro componentes que forman el sistema de predicción y control de carga de CoMo.

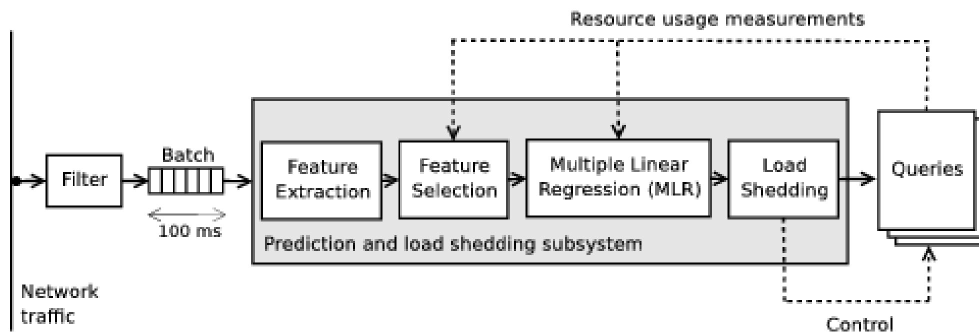


Figura 5-3 Funcionamiento del módulo de gestión de recursos de CoMo

El sistema de monitorización CoMo captura el tráfico de la red y agrupa los paquetes en conjuntos de 100 milisegundos. A continuación se realiza el cálculo de las diferentes features de cada conjunto (Feature Extraction) y se identifican aquellas que modelan mejor el uso de recursos de cada petición (Feature Selection). A partir de las features seleccionadas y basándose en observaciones anteriores del uso de recursos, el sistema predice el número de ciclos de CPU que necesitarán las peticiones de cada módulo para procesar el conjunto de paquetes (Multiple Linear Regresion). Si la predicción excede la capacidad del sistema, el subsistema de Load Shedding descarta una porción de paquetes, ya sea mediante muestreo de paquetes o de flujos, para reducir la carga del sistema. Finalmente, cada petición procesa el conjunto de paquetes resultante y se mide el consumo de CPU actual para poder utilizarlo en posteriores predicciones.

Una descripción más detallada de este sistema se puede encontrar en el artículo *Load Shedding in Network Monitoring Applications* [20].

5.4 Evaluación de las nuevas features basadas en la entropía

El principal objetivo de este capítulo es el comprobar si el uso de nuevas features basadas en el valor de la entropía puede mejorar el sistema de gestión de recursos de la herramienta CoMo. Para ello, se ha comprobado si mediante estas nuevas features el mecanismo de Load Shedding obtiene mejores predicciones del consumo de CPU que con las features originales y ver bajo

que condiciones unas se comportan mejor que otras. La versión de CoMo utilizada para realizar todos los análisis ha sido la versión 2.0.

La evaluación se ha realizado con dos configuraciones diferentes según el número de módulos que CoMo tiene activos:

- Solo un módulo activo correspondiente al contador de flujos IP (flowcount).
- Cinco módulos activos, explicados en la Tabla 5-1:

	Módulo	Descripción
1	Flowcount	Contabiliza el número de flujos IP presentes en la traza
2	Traffic	Contabiliza el número de paquetes y la cantidad de bytes
3	Protocol	Muestra el número de paquetes y bytes por protocolo presente en la traza
4	Topports	Muestra los puertos más populares con la cantidad de bytes como criterio
5	Tuple	Almacena los flujos de datos y el número de bytes de cada uno de ellos

Tabla 5-1 Módulos activos para la evaluación

También se ha evaluado el aumento del consumo de CPU del sistema de Load Shedding debido al uso de estas nuevas features, sin embargo los algoritmos desarrollados para la estimación de la entropía no han sido totalmente optimizados por lo que los resultados serán un peor caso que se podría mejorar considerablemente. El motivo de esta evaluación es que puede suceder que a pesar de obtener mejores predicciones, el costo de calcular las nuevas features haga que no sea rentable el utilizarlas. Es por ello que se ha evaluado el incremento del número de ciclos de CPU cuando se añaden las nuevas features al predictor respecto al coste del módulo de Load Shedding cuando solo se utilizan las features originales.

Las features añadidas al módulo de Load Shedding son las presentadas en el capítulo 3, las cuales consisten en el valor de la entropía de las combinaciones de campos de la cabecera TCP/IP mostradas en la Tabla 5-2:

Nº	Entropía de la distribución
1	IP origen
2	IP destino
3	IP origen + IP destino
4	Protocolo + Puerto origen
5	Protocolo + Puerto destino
6	Protocolo + IP origen + Puerto origen
7	Protocolo + IP destino + Puerto destino
8	Protocolo + Puerto origen + Puerto destino
9	5 - Tuple

Tabla 5-2 Nuevas features

Mediante el generador de anomalías desarrollado (explicado en el capítulo 4), también se ha comprobado la respuesta de dichas features en presencia de distribuciones del tráfico de red

anómalas, tales como ataques de denegación de servicio o escaneo de puertos. Además, también se ha comprobado como se comporta el sistema ante dos situaciones donde la entropía no está correlada con las features originales.

Por último, destacar que para la evaluación de las nuevas features se ha utilizado una nueva traza. El equipo de captura de esta traza se encuentra instalado en un centro de datos Equinix en Chicago y está conectado a un backbone OC192 de un ISP entre Chicago y Seatle. La traza fue capturada el 15 de enero de 2009 a las 13:59:15 UTC con una duración de cuarenta y cinco segundos.

5.4.1 Error de predicción

Para que el módulo de gestión de recursos Load Shedding funcione correctamente, este debe ser capaz de realizar buenas predicciones del consumo de CPU. Por este motivo, es importante evaluar la capacidad que tienen estas nuevas features para proporcionar buenas predicciones y así evitar una posible degradación del funcionamiento de CoMo. Para ello, se ha modificado el módulo de Load Shedding para que utilice únicamente las nuevas features basadas en la entropía, de modo que se puedan evaluar correctamente en ausencia del resto.

A continuación se muestra el error de predicción del módulo de Load Shedding a lo largo de la traza capturada utilizando únicamente las nuevas features como predictores. La Figura 5-4 muestra los resultados obtenidos cuando está activo únicamente el módulo flowcount y cuando están activos los 5 módulos.

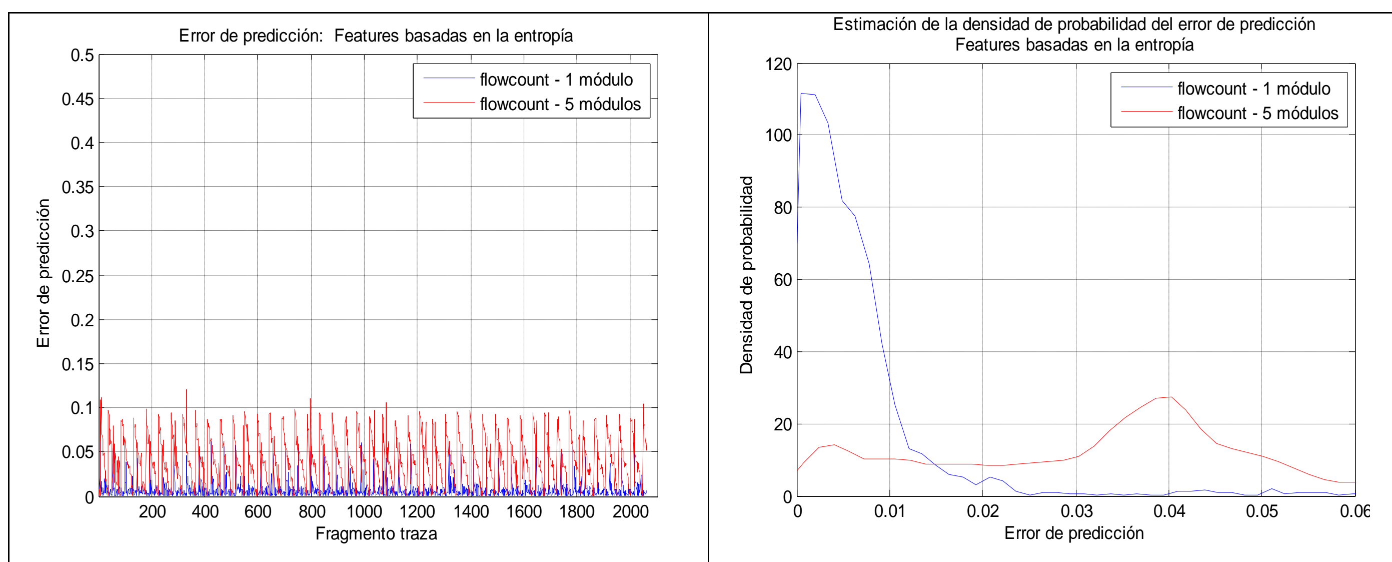


Figura 5-4 Error del predictor de Load Shedding- Features basadas en la entropía

En la Figura 5-4 se puede observar que las nuevas features proporcionan buenas predicciones del consumo de CPU a lo largo de la traza, donde el error medio es inferior al 1% cuando hay

solo un módulo activo, y de alrededor del 4% cuando están los 5 módulos. La figura de la derecha se observa una estimación obtenida mediante Matlab de la densidad de probabilidad del error de predicción que pretende mostrar la distribución del error. En esta se observa que el uso de más módulos incrementa la media y la varianza del error de predicción.

Por otra parte se ha comprobado cual es el comportamiento del sistema cuando se utilizan todas las features, es decir, las 55 features antiguas y las 9 nuevas, respecto al funcionamiento original del módulo de Load Shedding. En la Figura 5-5 se muestra una estimación de la distribución del error de predicción a partir de los datos obtenidos con la configuración original, con las nuevas features, y con la combinación todas estas features para el módulo flowcount.

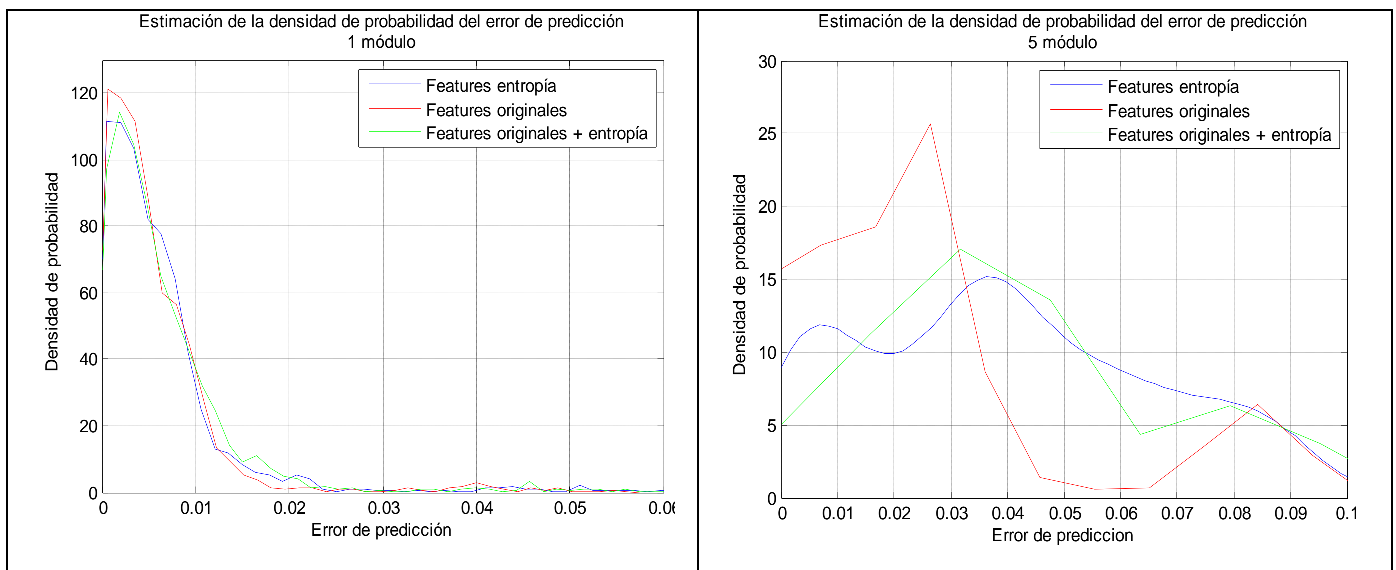


Figura 5-5 Error de predicción en función de las features utilizadas - flowcount

La Figura 5-5 muestra que con la traza actual apenas hay diferencia utilizando las nuevas features cuando solo hay un módulo activo, y en el caso de los 5 módulos activo, aumenta la desviación estándar del error. Estos resultados son para el caso del módulo flowcount, pero el comportamiento es similar cuando están activos el resto de módulos en solitario.

Como se ha indicado, estos resultados se han obtenido cuando CoMo está trabajando con un solo módulo activo, lo que implica que no esté consumiendo muchos recursos del procesador. Por este motivo, se ha evaluado la situación en que CoMo trabaja con los 5 módulos descritos en la Tabla 5-1 activados y realizando predicciones del consumo de recursos para cada uno de ellos.

A continuación se presentan los resultados obtenidos de forma gráfica donde se ha muestra la media y la desviación estándar del error de predicción de cada módulo para las tres situaciones presentadas anteriormente con los 5 módulos funcionando en paralelo.

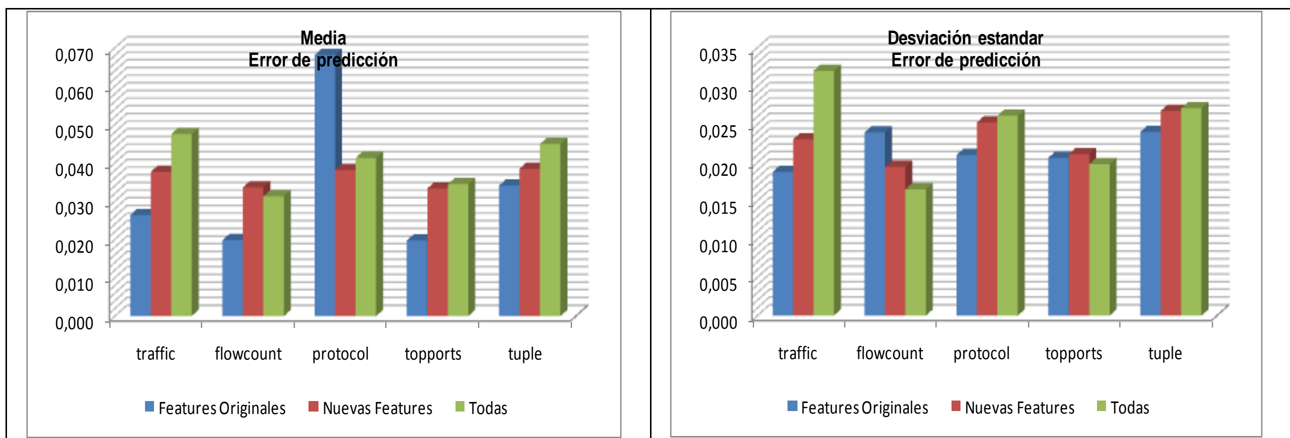


Figura 5-6 Media y desviación estándar del error de predicción de los 5 módulos

La Figura 5-6 muestra que en una captura de tráfico real de un ISP las nuevas features no mejoran las predicciones obtenidas con las features originales. Esto se debe a que la feature de la entropía está pensada para tráfico con anomalías donde la distribución de la información de los paquetes suele ser diferente a cuando no hay. Además, la entropía en estos casos está muy correlada con el resto de features así que no aporta nueva información, pudiendo confundir al predictor. Por este motivo, se puede observar que en todos los módulos el error ha aumentado al utilizar la entropía.

5.4.2 Consumo de las nuevas features

La evaluación del coste de las nuevas features se ha realizado para poder comprobar si su uso es viable o si por el contrario su estimación utilizará demasiados ciclos de CPU. Para ello se ha analizado la traza de estudio sin las nuevas features y con ellas y se ha observado el valor del coste de ciclos del módulo de Load Shedding.

Es importante destacar que los algoritmos desarrollados no han sido optimizados mientras que las features actuales se calculan mediante unos algoritmos muy eficientes. Esto implica que los resultados se pueden considerar un peor caso y podrían mejorarse considerablemente si se implementase las optimizaciones adecuadas.

En la Figura 5-7 se puede observar el incremento del consumo del módulo de Load Shedding cuando se utilizan las nuevas features respecto a cuando solo se utilizan las features originales en forma de porcentaje. La figura muestra este incremento a lo largo del tiempo cuando hay un módulo activo y cuando están los 5 módulos activos.

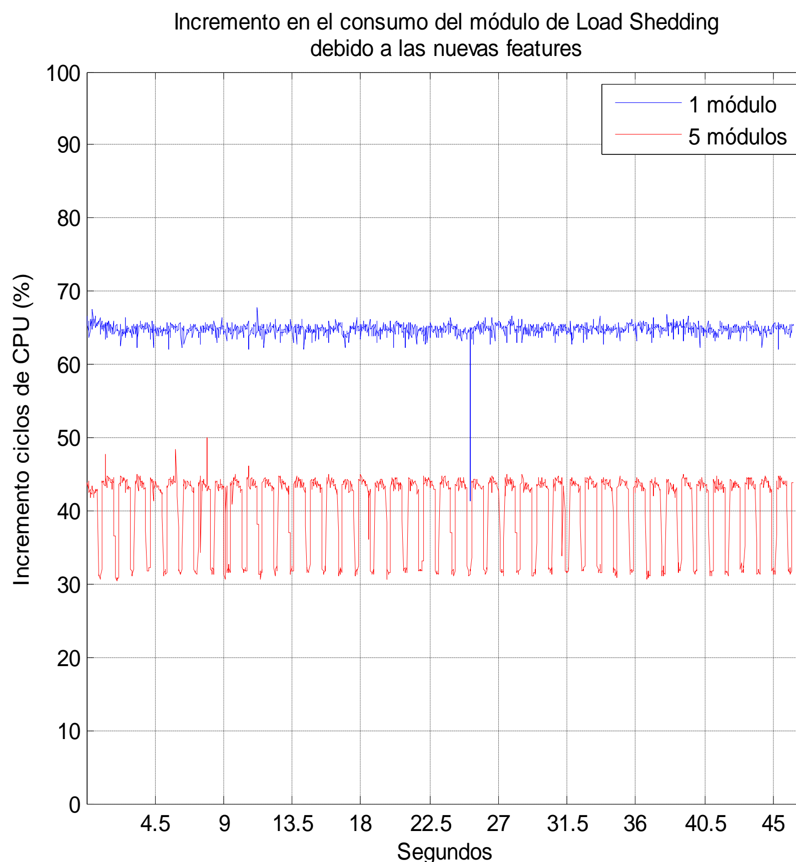


Figura 5-7 Porcentaje de aumento de consumo de Load Shedding con las nuevas features

La figura anterior muestra que el consumo de ciclos de CPU necesarios con un módulo activo y cuando se utiliza las nuevas features aumenta del orden del 60% respecto a cuando no se utilizan estas features, y de alrededor del 40% cuando están los 5 módulos activos. En el primer caso, un aumento del 60% es excesivo y se debería considerar la posibilidad de no utilizar las nuevas features. Sin embargo, en el caso de que haya múltiples módulos, el aumento del consumo de CPU del módulo de Load Shedding se podría soportar.

Además, se ha de tener muy presente en esta evaluación que los algoritmos probados no han sido nada optimizados por lo que se podría reducir considerablemente estos valores, estimando una reducción de aproximadamente el 50%, sin olvidar que las features originales se calculan con algoritmos especializados que utilizan muy pocos recursos.

5.4.3 Comportamiento de las features en presencia de anomalías

Un último análisis que se ha querido realizar es el comportamiento de las nuevas features ante ataques o anomalías en el tráfico de red capturado. En nuestro caso se ha utilizado la misma traza de tráfico que en los estudios anteriores, pero se le han añadido tres tipos de anomalías diferentes.

Las anomalías insertadas en la traza de estudio son un ataque de denegación de servicio distribuido, un escaneo de puerto de redes en busca de servidores web activos (puerto 80) y un

anomalía donde la entropía toma valores extremos próximo a 0 y 1 Estas anomalías se han añadido mediante el software desarrollado en el presente proyecto de final de carrera y cuyo funcionamiento se encuentra explicado en el capítulo 4 de esta memoria.

5.4.3.1 DDoS

La primera anomalía evaluada ha sido un ataque de denegación de servicio distribuido cuya finalidad es la de inhabilitar a un servidor para que no pueda ofrecer un determinado servicio a sus usuarios. Este ataque se caracteriza por un gran número de paquetes IP con destino a una sola máquina y a un puerto determinado. En concreto, se han realizado multitud de intentos de conexiones TCP con la petición de conexión (SYN) y la confirmación (ACK) del servidor correspondiente al puerto TCP 80, simulando un ataque contra un servidor de páginas web. Además, el hecho de que sea distribuido quiere decir que existen múltiples atacantes con direcciones IP diferentes por lo que existirán muchas direcciones IP origen diferentes. El modelo de la anomalía se puede ver en el anexo B.1.

Aprovechando el desarrollo de las nuevas features, a continuación se muestra la entropía de la traza de tráfico evaluada antes y después de añadir la anomalía, para las características que se ven modificas.

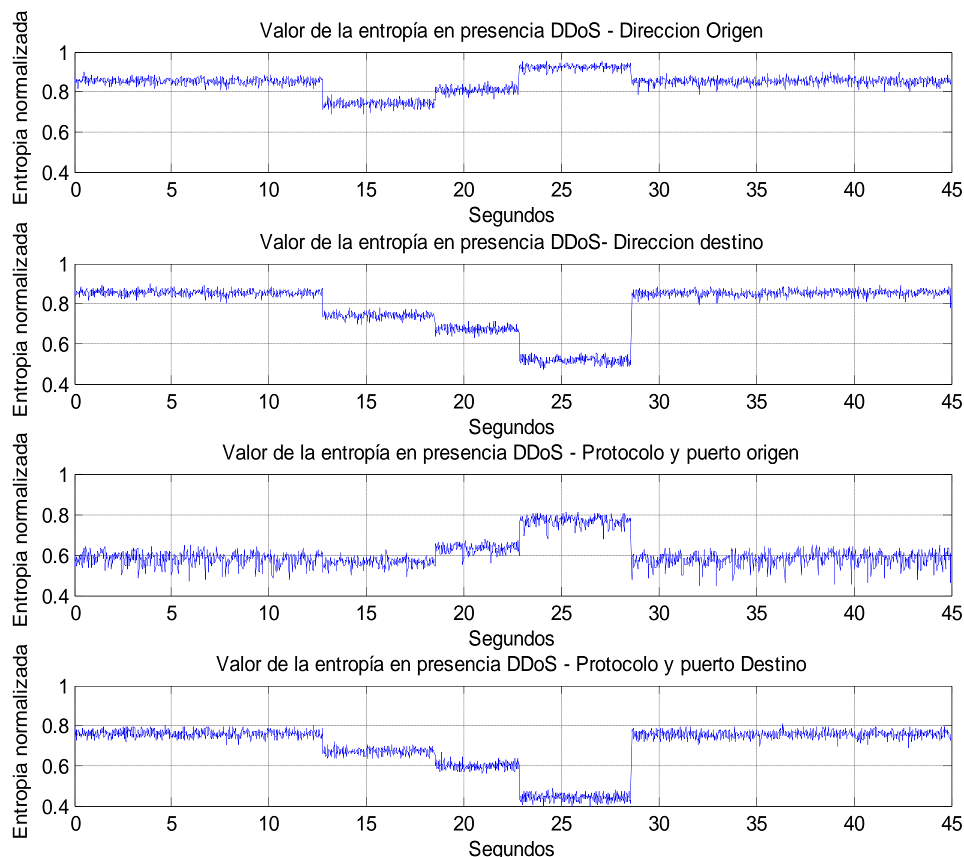


Figura 5-8 Entropía de las direcciones origen, destino y puertos origen y destino

En la Figura 5-8 se puede ver la duración de la anomalía, aproximadamente 15 segundos, y como varía el valor de la entropía para las cuatro características que se ven más afectadas. Además, se pueden apreciar claramente tres tramos, los cuales se explican de la siguiente forma: durante el primer tramo el servidor atacado es capaz de responder a todas las peticiones de conexión; durante el segundo tramo el servidor empieza a quedarse sin recursos y ya no es capaz de servir todas las peticiones pero sigue respondiendo a un gran número de ellas; el tercer tramo simula que el servidor ya no da abasto y ha dejado de servir cualquier tipo de petición.

Por ejemplo, si nos fijamos en la entropía de la distribución de las direcciones origen, vemos que en el primer tramo se reduce un poco debido a que a pesar de tratarse de un ataque distribuido, se concentra en una sola dirección IP y esta genera siempre un paquete de respuesta (ACK) por lo que el desorden de esta variable disminuye. Sin embargo, a medida que va pasando el tiempo y el servidor deja de responder, la aleatoriedad del origen del ataque hace que la entropía vaya aumentado hasta que alcanza los valores máximos cuando el servidor atacado deja de responder y, por tanto, solo quedan en la traza los paquetes con la dirección origen aleatoria y los paquetes existentes en la traza capturada.

El comportamiento del valor de la entropía de las direcciones destino es el opuesto al caso anterior, donde en el tercer tramo todos los paquetes se concentran en la dirección IP del servidor atacado y como este no responde, el valor de la entropía disminuye al haber menos variedad.

La Figura 5-9 muestra la evolución de algunas de las features originales en comparación con algunas de las nuevas features basadas en el valor de la entropía:

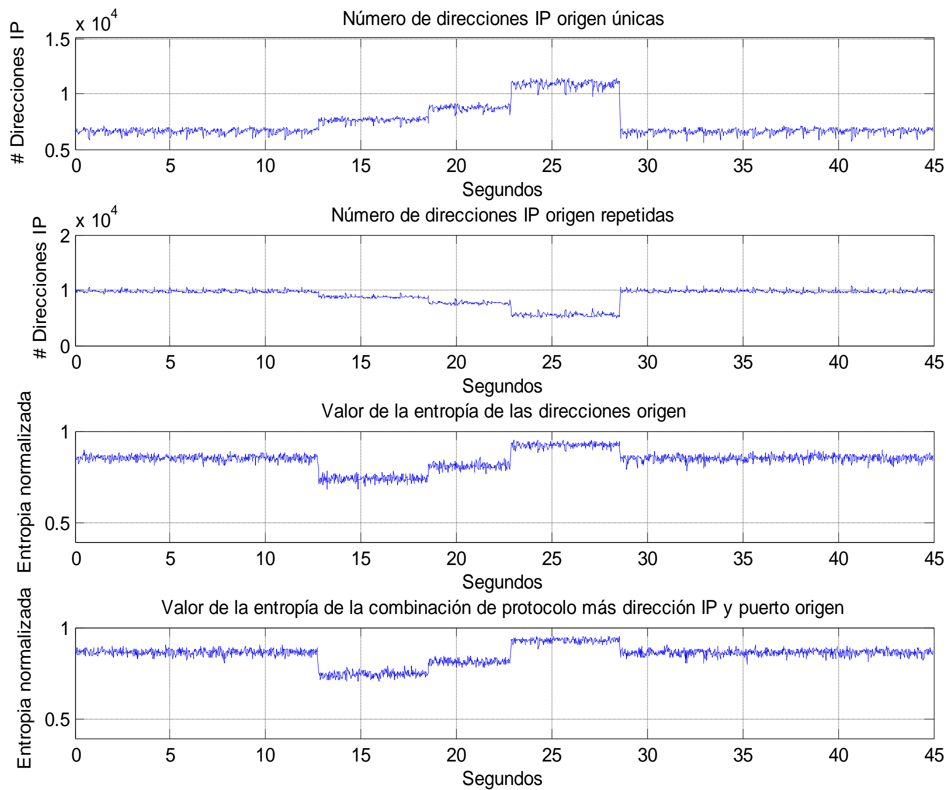


Figura 5-9 Entropía de las direcciones origen, destino y puertos origen y destino

En esta figura se puede observar que existe una gran correlación entre las diferentes features lo que provocará que la posible mejora de las nuevas features no se vea reflejada en los resultados.

La Figura 5-10 muestra la distribución de la densidad del error de predicción para los casos de uso de las features originales o las features originales más las nuevas features durante el período de tiempo que está presente la anomalía. Además se comparan las situaciones en que hay un módulo activo (flowcount) o varios módulos activos.

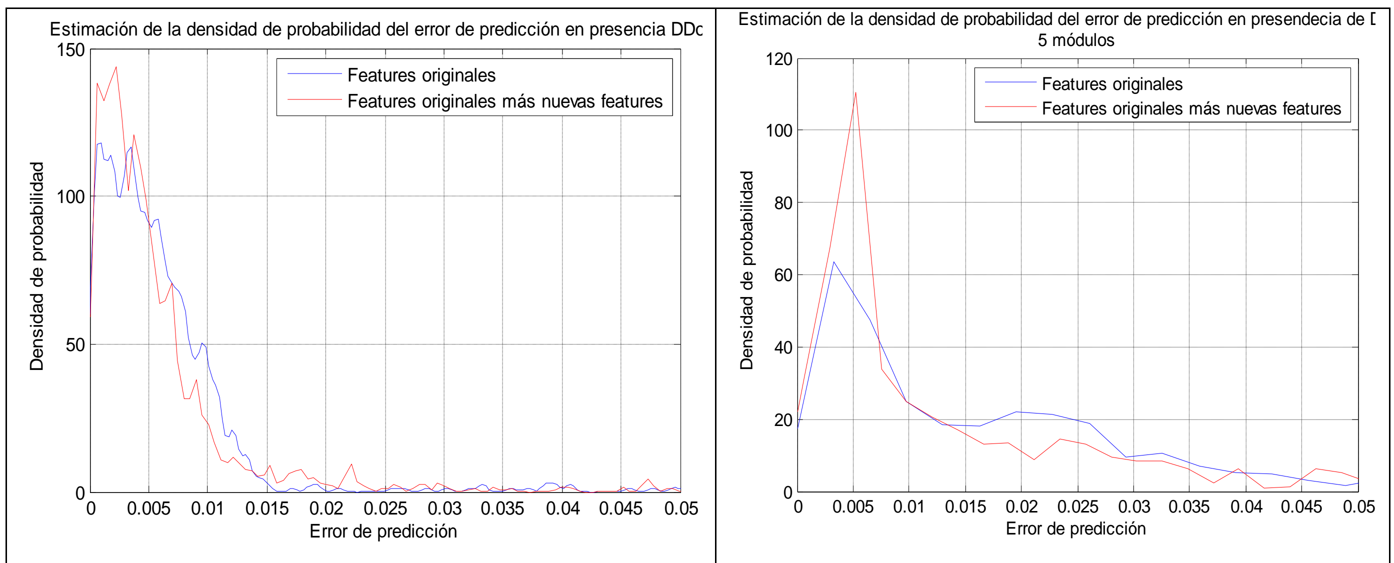


Figura 5-10 Error de predicción en presencia de un ataque DDoS con 1 módulo activo

En la figura anterior se puede ver que el comportamiento del sistema con las nuevas features mejora muy poco, debido principalmente a la correlación entre las features originales y las nuevas y a que no hay mucho margen de mejora en el error de predicción al estar por debajo del 1%.

A continuación se presenta la Figura 5-11 con la media y la desviación estándar en las tres configuraciones planteadas: con las features originales, con las nuevas features y con la combinación de las originales y las nuevas durante el periodo en que está presente la anomalía. En este caso, se muestra el resumen de los resultados obtenidos cuando los 5 módulos están activos.

Figura 5-11 Error de predicción en presencia de un ataque DDoS con 5 módulos activos

En esta figura se puede ver claramente que la precisión del estimador varía mucho en función del módulo que se evalúe. Por ejemplo, el módulo que tiene un menor error medio es el *flowcount*, con errores por debajo del 1%, sin embargo el módulo *tuple* tiene un error mucho mayor, del orden del 10%. Esta figura también muestra que no existe mejora utilizando las nuevas features debido a que estas no son seleccionadas por el módulo correspondiente de Load Shedding, ya que están correladas con las features originales y no aportan nueva información.

5.4.3.2 NetScan

El propósito de este segundo tipo de anomalía es el de localizar dentro de una red si existe un servicio determinado corriendo en una equipo. En este caso, la simulación de la anomalía consiste en generar múltiples paquetes IP desde un origen fijo a múltiples destinos dentro de un rango de direcciones IP y un puerto fijo. En la anomalía generada, se utiliza el puerto 80 para simular que se está buscando un servidor web, aunque se podría buscar cualquier otro servicio de los denominados *well-known ports* [22] (servicios habituales cuyo puerto es conocido).

A continuación, la Figura 5-12 muestra el valor de la entropía a lo largo del tiempo para las diferentes características del tráfico de red afectadas por la anomalía.

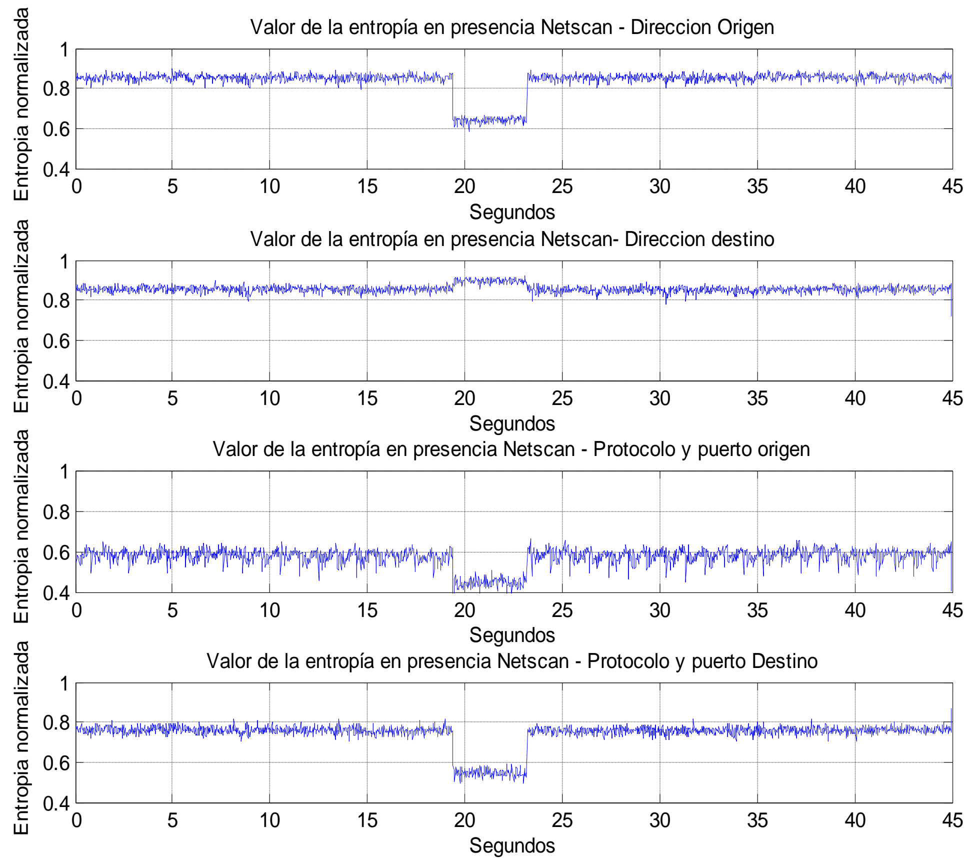


Figura 5-12 Entropía de las direcciones origen, destino y puertos origen y destino

En esta figura se puede observar que la duración de la anomalía es de 5 segundos que corresponde al barrido del puerto 80 de todas las direcciones IP pertenecientes a la red 17.43.0.0/16. En este caso, el valor de la entropía de la dirección destino aumenta al estar mirando múltiples direcciones, mientras que la dirección origen es fija y por este motivo durante la anomalía el valor de la entropía es menor que en el resto de la traza. El modelo de la anomalía se puede ver en el anexo B.2.

Respecto al comportamiento del módulo de gestión de recursos, en la Figura 5-13 se puede ver la distribución de la densidad de probabilidad del error de predicción durante la anomalía para el módulo flowcount.

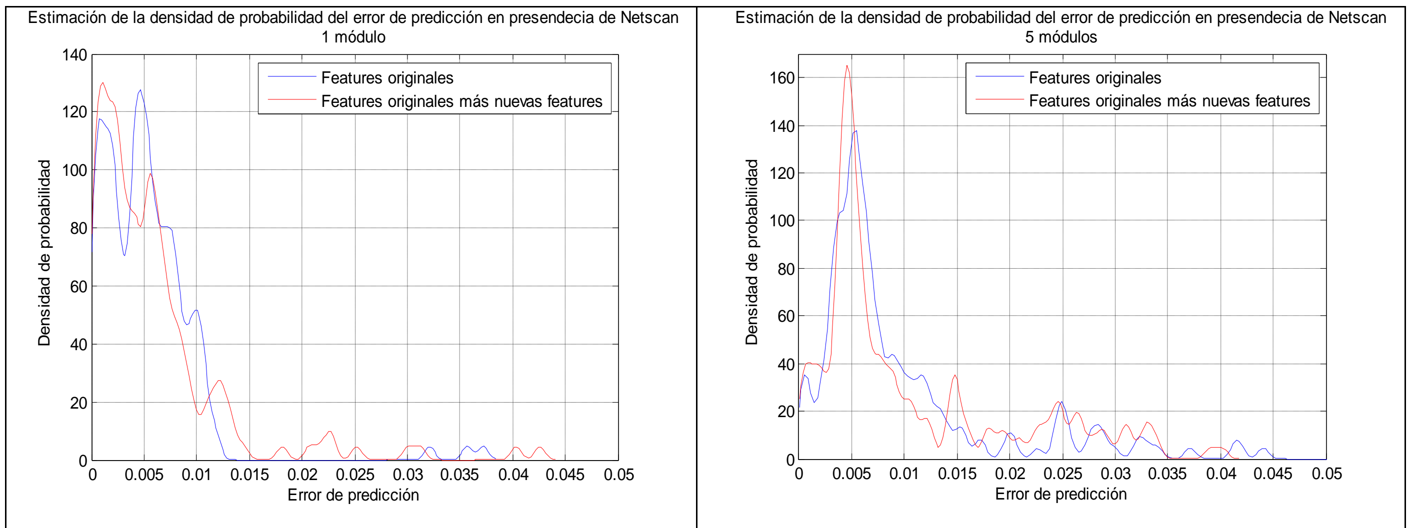


Figura 5-13 Error de predicción en presencia de un ataque NetScan con 1 módulo activo

Al igual que para la anomalía de denegación de servicio, los resultados obtenidos son muy parecidos cuando se utilizan las nuevas features y cuando no. De igual forma, el error de predicción en ambos casos suelen estar por debajo del 1% así que tampoco existe mucho margen de mejora con las nuevas features.

El mismo análisis se ha realizado para el caso en que CoMo tenga activos 5 módulos. En la Figura 5-14 se muestra el error medio y la desviación estándar para cada uno de los módulos en los 3 escenarios de uso de features planteados. Estos datos corresponden al período en que la anomalía está presente en la traza.

Figura 5-14 Error de predicción en presencia de un ataque Netscan con 5 módulos activos

Los resultados muestran que el error medio de predicción al usar las nuevas features en general tampoco mejora por el mismo motivo, las nuevas features no aportan nueva información al predictor. A pesar de ello, se observa una ligera mejora tanto en el módulo protocol como tuple.

5.4.3.3 *Mínimo y máximo de entropía*

Este tipo de anomalía pretende representar un ataque a determinadas herramientas que se basan en localizar patrones en los primeros paquetes de un flujo de datos. Para ello, se ha modificado el módulo *tuple* para que realice una operación costosa durante los primeros 100 paquetes de cada flujo y de esta forma, si la distribución de flujos varia, el coste de la ejecución del módulo también variará.

En este caso la anomalía consiste en dos fases de 1 segundo cada una repetidas 10 veces, para mostrar mejor el comportamiento del predictor. Durante cada fase se generan flujos con una distribución tal que se presentan máximos y mínimos del valor de la entropía de los flujos en cada fase. La diferencia entre cada fase será que en unos casos se realizará la operación las 100 veces para todos los flujos y en la otra fase solo se realizará las 100 veces para uno determinado, mientras el resto de flujos aparecerán pocas veces y la operación no llegará a hacerse las 100 veces. El modelo de la anomalía se puede ver en el anexo B.3.

En la Figura 5-15 se puede observar el valor de distintas features a lo largo de la anomalía en comparación con la features originales, mostrando claramente que en este caso la entropía no está correlada con las features originales.

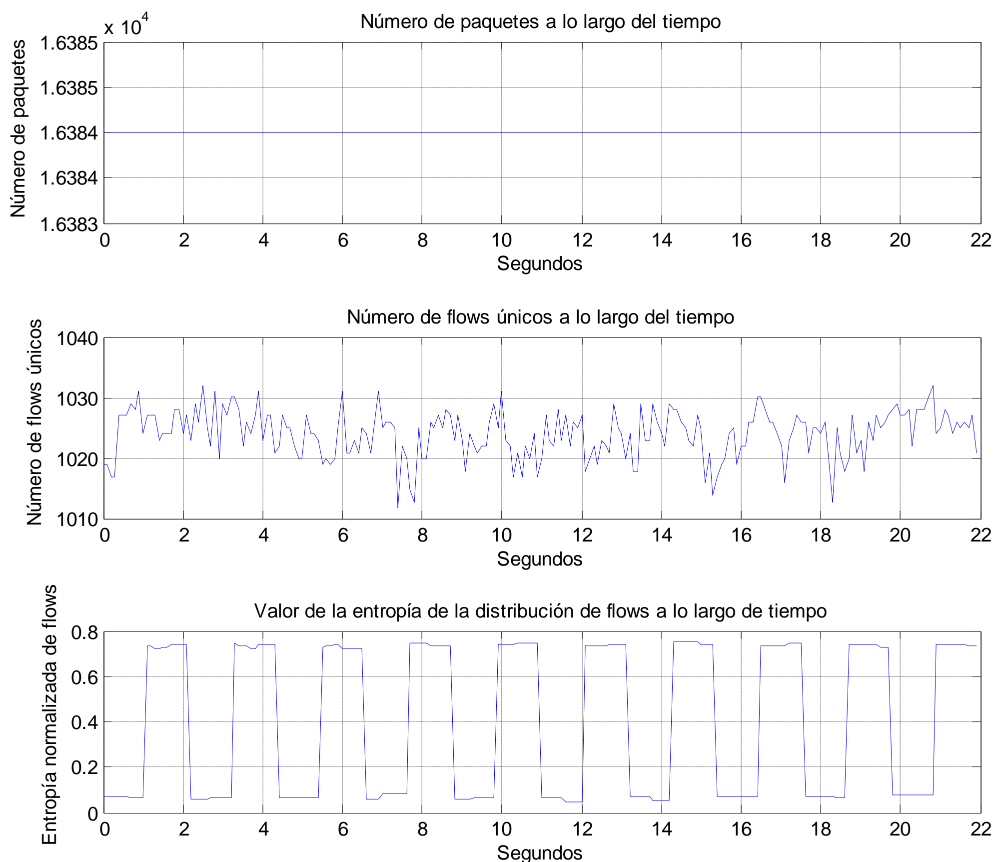


Figura 5-15 Valor de varias features durante la nueva anomalía

Se puede ver claramente las dos fases mencionadas en el valor de la entropía de la distribución de los flujos, y como esta feature es capaz de caracterizar esta situación anómala a diferencia de las features originales. En la figura se ve que ni el número de paquetes ni el número de flujos únicos varía a lo largo de la anomalía, al igual que el resto de features originales, mientras que su distribución, caracterizada por el valor de la entropía, si que lo hace.

A continuación, la Figura 5-16 muestra el error de predicción del módulo de Load Shedding cuando se utilizan las features originales, cuando se utilizan solo las nuevas features y cuando se utilizan la combinación de ambas.

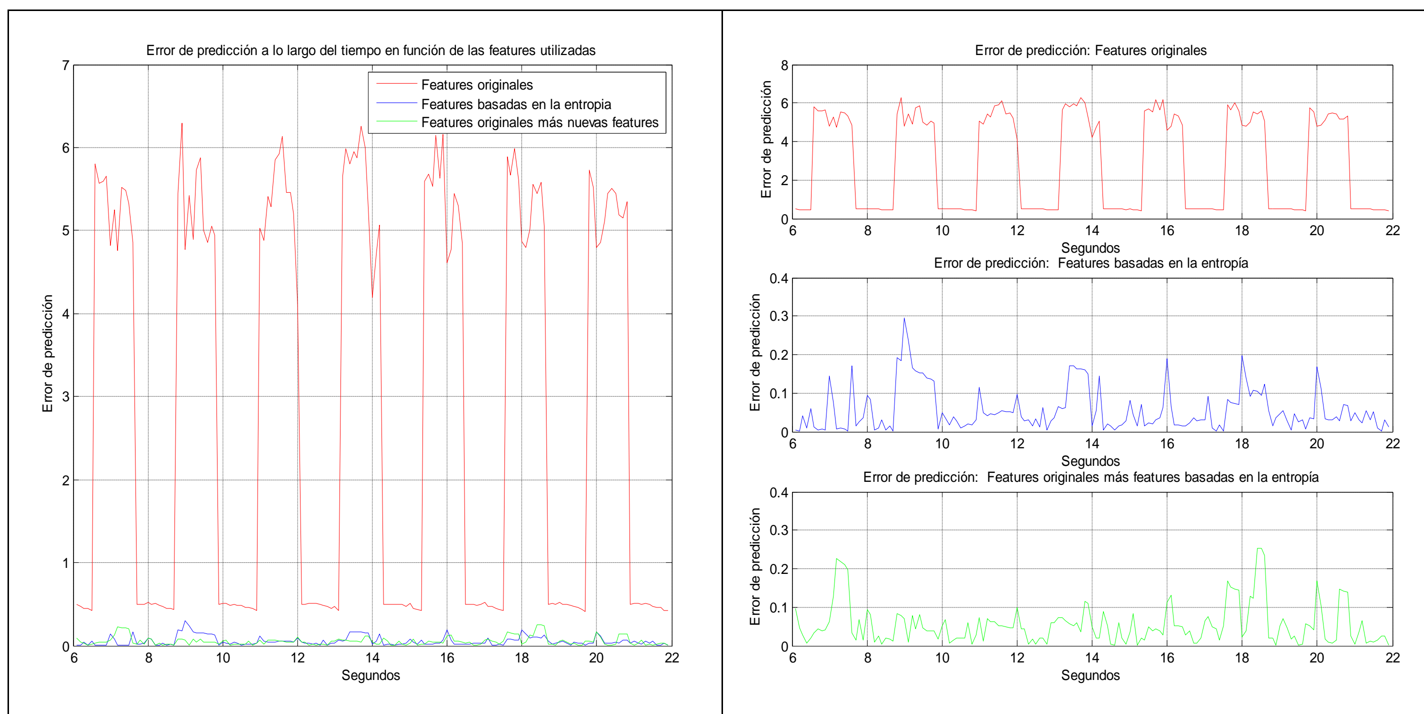


Figura 5-16 Error de predicción en presencia de la anomalía: módulo tuple modificado

A diferencia de las dos anomalías anteriores, en este caso se puede ver la mejora al introducir las nuevas features basadas en la entropía. En este caso, se observa que las features originales son incapaces de predecir el coste del módulo tuple cuando están presentes ambas fases de la anomalía. Sin embargo, solo utilizando las nuevas features ya se obtienen muy buenas predicciones con alrededor del 5% de error, mientras que combinando todas las features se obtiene una ligera mejora respecto al caso anterior.

5.5 Conclusiones

Las features originales obtienen muy buenas predicciones en situaciones normales aunque es difícil saber si serán capaces de ser útiles en circunstancias anómalas. Por este motivo, el uso de las nuevas features pretende aportar información que hasta ahora el módulo de Load Shedding

no es capaz de obtener, permitiendo abarcar situaciones en las que hasta ahora no se disponía de suficiente información.

Sin embargo, estas situaciones son difíciles de simular ya que el acceso a tráfico real es complejo por cuestiones de privacidad y medios, y porque resulta muy difícil el realizar simulaciones de la gran cantidad de estadísticas de tráfico diferentes que se pueden encontrar en las redes de un proveedor de internet y la mayor sofisticación de los nuevos ataques que van apareciendo.

Los resultados obtenidos muestran que en ausencia de anomalías, los errores de predicción obtenidos con el uso de las nuevas features son similares a los obtenidos por las features originales. El principal motivo de esto es que el valor de la entropía apenas varía a lo largo de la traza capturada por lo que no afectará a la estimación del consumo de CPU de los diferentes módulos. Además, la correlación entre las features originales y las nuevas es muy alta en estas situaciones por lo que las nuevas features no proporcionan nueva información.

Por otro lado, en las pruebas en presencia de las dos primeras anomalías se puede ver que las nuevas features no mejoran los resultados obtenidos, lo que puede llevar a pensar que quizás no sean necesarias. Esto se debe a que las features originales y las nuevas siguen estando correladas por lo que la mejora que se puede obtener es inapreciable. Sin embargo, en la simulación de la tercera anomalía hemos obtenido unos resultados muy buenos. En este caso, se ha demostrado que las features originales son incapaces de realizar predicciones del consumo de CPU de cada módulo, mientras que las nuevas features obtienen errores pequeños. Cabe destacar que este escenario no es improbable y tanto el módulo utilizado como las anomalías son realistas, por lo que la estimación de la entropía mejora el sistema de gestión de recursos de CoMo frente a anomalías.

Por otra parte, se ha comprobado que la estimación de la entropía supone un incremento del coste de CPU al añadir nuevas features. El uso de estas nuevas features ha supuesto un aumento del 40% de sobrecarga asociado al módulo de Load Shedding que, teniendo en cuenta que los algoritmos no han sido optimizados, se puede considerar que no es crítico. Además, el cálculo de las features originales se realiza de una forma extremadamente eficiente, por lo que en comparación, el coste de las nuevas features parece mucho mayor. Por último, se podría reducir el coste total del módulo de Load Shedding sustituyendo algunas de las features originales por las nuevas features.

Capítulo 6

Planificación y estudio económico

6.1 Planificación

El desarrollo de este proyecto se ha dividido en tres fases, cada una dividida en diferentes etapas. A continuación se detallan cada una de estas fases, indicando la categoría profesional del encargado de llevar a acabo cada una de ellas.

1. Fase de definición de proyecto (jefe de proyecto y analista)

- Organización inicial de las tareas a realizar (jefe de proyecto)
- Investigación de la situación actual respecto a herramientas de monitorización, algoritmos de estimación de la entropía y generación e inserción de tráfico sobre una traza de red (analista)
- Estudio de los algoritmos de estimación de entropía a desarrollar (analista)
- Estudio de CoMo (analista)
- Estudio de Flame y de la literatura relacionado con anomalías en redes IP (analista)
- Estudio del módulo de gestión de recursos Load Shedding (analista)
- Planificación del proyecto (jefe de proyecto y analista)

2. Fase de desarrollo (analista i programador)

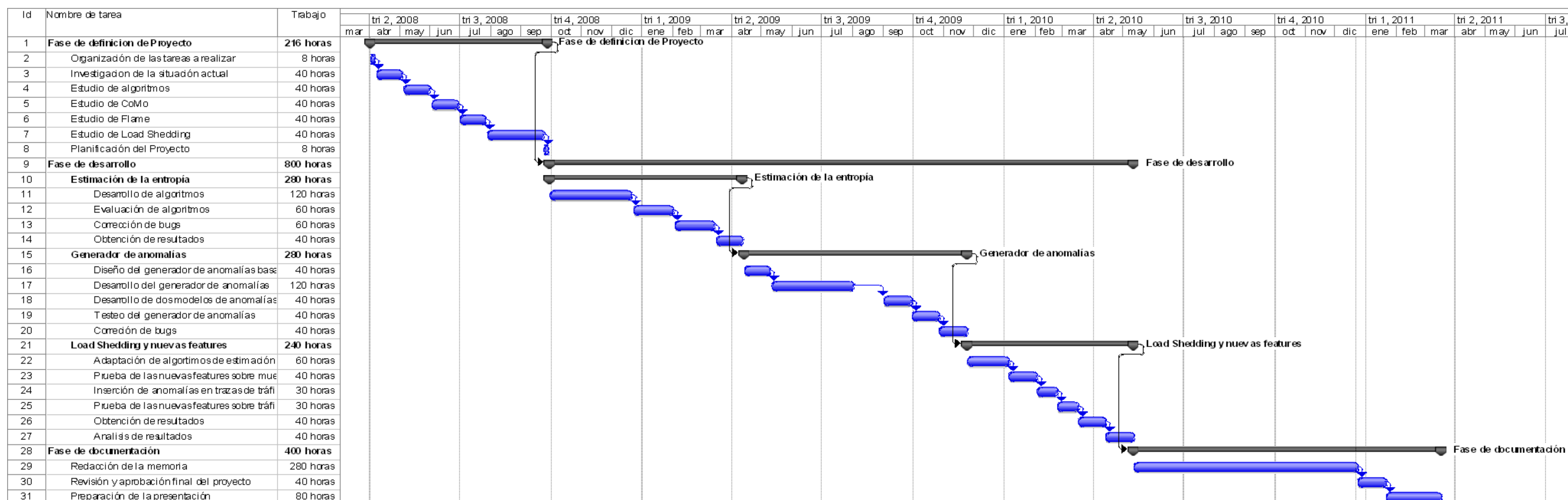
- Estimación de la entropía
  - Desarrollo de los algoritmos de estimación de entropía (programador)
  - Evaluación de los algoritmos y obtención de resultados con varias trazas de tráfico de red reales (programador y analista)
- Generador de anomalías
  - Diseño del generador de anomalías basado en FLAME y de varios modelos de anomalías (analista)

  - o Desarrollo del generador de anomalías y de los modelos de anomalías seleccionados (programador)
  - o Testeo del generador de anomalías y corrección de errores (programador y analista)
- Load Shedding y uso de la entropía
  - o Adaptación de los algoritmos de estimación de la entropía al módulo de Load Shedding (analista)
  - o Prueba de las nuevas features sobre muestras de tráfico real (programador)
  - o Inserción de anomalías en trazas de tráfico real (programador)
  - o Prueba de las nuevas features sobre tráfico con anomalías (programador)

3. Fase de documentación (jefe de proyecto y analista)

- Redacción de la memoria definitiva (analista)
- Revisión y aprobación final del proyecto (jefe de proyecto y analista)
- Preparación de la presentación (jefe de proyecto y analista)

En la Figura 6-1 se puede ver una planificación temporal del proyecto con una dedicación aproximada de 10 horas semanales.



Id	Nombre de tarea	Trabajo
1	**Fase de definicion de Proyecto**	**216 horas**
2	Organización de las tareas a realizar	8 horas
3	Investigacion de la situación actual	40 horas
4	Estudio de algoritmos	40 horas
5	Estudio de CoMo	40 horas
6	Estudio de Flame	40 horas
7	Estudio de Load Shedding	40 horas
8	Planificación del Proyecto	8 horas
9	**Fase de desarrollo**	**800 horas**
10	**Estimación de la entropía**	**280 horas**
11	Desarrollo de algoritmos	120 horas
12	Evaluación de algoritmos	60 horas
13	Correción de bugs	60 horas
14	Obtención de resultados	40 horas
15	**Generador de anomalías**	**280 horas**
16	Diseño del generador de anomalías basa	40 horas
17	Desarrollo del generador de anomalías	120 horas
18	Desarrollo de dos modelos de anomalías	40 horas
19	Testeo del generador de anomalías	40 horas
20	Coreción de bugs	40 horas
21	**Load Shedding y nuevas features**	**240 horas**
22	Adaptación de algortimos de estimación	60 horas
23	Prueba de las nuevas features sobre mue	40 horas
24	Inserción de anomalías en trazas de tráfi	30 horas
25	Prueba de las nuevas features sobre tráfi	30 horas
26	Obtención de resultados	40 horas
27	Analisis de resultados	40 horas
28	**Fase de documentación**	**400 horas**
29	Redacción de la memoria	280 horas
30	Revisión y aprobación final del proyecto	40 horas
31	Preparación de la presentación	80 horas

Figura 6-1 Planificación tareas a realizar

.

6.2 Estudio económico

6.2.1 Presupuesto de recursos humanos

En la Tabla 6-1 se puede ver un resumen del número de horas dedicadas para cada una de las categorías profesionales, los sueldos estimados y el coste total del presupuesto de recursos humanos

Categoría	Sueldo/Hora	Horas	Sueldo total
Jefe de proyecto	35 €	56	1960 €
Analista	30 €	680	19500 €
Programador	20 €	440	11600 €
	Total cal	1176	33060 €

Tabla 6-1 Presupuesto de recursos humanos

El número total de horas planificadas para el analista y el programador son 1120 que aproximadamente corresponden a las 1080 horas asignadas a los 36 créditos del proyecto de final de carrera en la ETSETB.

6.2.2 Presupuesto de equipamiento informático

Para el desarrollo del presente proyecto no se ha requerido equipamiento informático específico para la ejecución de la herramienta de monitorización, de los algoritmos desarrollados y del generador de anomalías. Por este motivo el único equipamiento necesario ha sido el utilizado como entorno de desarrollo.

Para esta tarea se ha dispuesto de un portátil Dell XPS M1330 en el cual se han desarrollado los algoritmos y ejecutado la herramienta CoMo, y se ha desarrollado el generador de anomalías. Además todo el software utilizado en el presente proyecto es software libre, por lo que no se ha considerado en el presupuesto ningún coste de licencias de uso.

Para calcular el coste de los equipos informáticos se ha considerado que estos tienen una vida útil de 3 años y solo se ha contabilizado el periodo que se ha utilizado. En la Tabla 6.2 se describen estos costes.

Descripción	Tareas	Especificaciones	Unidades	Precio/Unidad	Tiempo de uso	Coste por uso
Dell XPS M1330	Desarrollo, evaluación y redacción del proyecto.	Intel Core 2 Duo T8100 4GB RAM 250GB Disco duro	1	1300	2 años	867 €

Tabla 6-2 Coste del equipamiento informático

6.2.3 Presupuesto general

El presupuesto general correspondiente al desarrollo de este proyecto es la suma del coste en recursos humanos y del coste en equipamiento, por lo que finalmente el presupuesto general asciende a 33927 €.

UPC
telecom BCN

Capítulo 7

Conclusiones finales y trabajos futuros

La aparición de nuevas tipos de ataques y de anomalías en Internet requiere la constante mejora y evolución de las herramientas de análisis y monitorización del tráfico que circula por la red. El uso de la entropía como entrada de información para estos sistemas se ha demostrado que es muy útil en caso de situaciones anómalas.

En este sentido, los algoritmos desarrollados en el presente proyecto han conseguido buenas estimaciones de la entropía reduciendo considerablemente el uso de recursos del sistema, tanto en espacio de memoria como en consumo de CPU, frente al caso del cálculo exacto de la entropía. En particular, se ha obtenido que ambos algoritmos son capaces de obtener buenas estimaciones del valor de la entropía, alcanzando errores relativos inferiores al 5% en la mayoría de las configuraciones y características del tráfico.

Respecto a la eficiencia de ambos algoritmos se ha comprobado que el algoritmo de Sieving es más eficiente que el de Stremaing en el contexto de uso en la herramienta CoMo. En particular, este algoritmo es capaz de proporcionar errores relativos inferiores al 5% utilizando solo el 10% de contadores respecto al algoritmo de cálculo exacto de la entropía. Sin embargo, el algoritmo de Streaming no ha demostrado ser tan eficiente lo que se debe principalmente al hecho de que el número de contadores utilizados en este algoritmo es logarítmicamente proporcional al tamaño del fragmento de tráfico de red del que se realiza la estimación, es decir, el número de contadores necesarios crece más lentamente a medida que aumenta el número de paquetes. En la práctica, CoMo utiliza fragmentos de 100 milisegundos de tráfico donde se han encontrado un máximo de 16384 paquetes en las trazas utilizadas. De este modo, se ha concluido que el algoritmo más adecuado para su uso en CoMo es el de Sieving.

A parte de esto, también se ha detectado que no tiene mucho sentido utilizar estos algoritmos para la estimación de la entropía de la distribución de protocolos en una traza de red debido a que el número de posibles valores para los protocolos es pequeño. Esto hace que el algoritmo de

Sieving se comporte de forma similar al algoritmo que realiza el cálculo exacto y por tanto no se consiga ningún beneficio, mientras que el algoritmo de Streaming queda totalmente desaconsejado al utilizar muchos más contadores de forma innecesaria.

Por último, se ha comprobado que ambos algoritmos reducen el coste computacional para estimar la entropía, pero esta reducción no es muy significativa en la fase de cálculo de contadores ya que el algoritmo es similar que el del cálculo del valor exacto de la entropía. Sin embargo, donde si que hay una buena optimización es en la fase de post-procesado ya que ésta depende directamente del número de contadores necesarios, y como ya se ha indicado, ambos algoritmos son capaces de reducir su número.

Respecto a los posibles trabajos futuros, actualmente se utilizan tablas de hash para el almacenamiento de los contadores necesarios para estimar la entropía. Sin embargo, las features actuales de CoMo hacen uso de los algoritmos de bitmaps de múltiple resolución [23] que realizan el cálculo de contadores de una forma muy eficiente, por lo que sería interesante considerar si técnicas similares se pueden aplicar a la estimación de la entropía.

En cuanto al desarrollo del generador de anomalías podemos concluir que se trata de una herramienta que puede ser útil en un entorno académico que requiera el uso de trazas con presencia de tráfico anómalo. Se ha conseguido mantener la flexibilidad y sencillez de la herramienta FLAME y se han añadido funciones como la modificación de tráfico existente. Además, actualmente este software ya está siendo utilizado por los estudiantes de postgrado del CCABA del Departamento de Arquitectura de Computadores de la UPC. En este sentido, el número de ataques en Internet crece día a día y el uso de este tipo de herramientas será cada vez más útil para analizar la realidad del tráfico que circula diariamente por Internet.

En un futuro sería interesante ampliar funciones del generador, como la posibilidad de duplicar paquetes de la traza de entrada o aceptar múltiples fuentes de datos. Otra opción interesante es el diseño de paquetes con otros protocolos que no sean TCP o UDP. En este sentido, existe un desarrollo de software libre denominado DPKT [24] que permite la creación de paquetes de multitud de protocolos diferentes y está desarrollado en Python por lo que en principio no debería ser muy compleja su integración con el generador de anomalías desarrollado.

Por último, sea han obtenido los resultados esperados en la evaluación del uso de la entropía como métrica para el módulo de gestión de recursos de Load Shedding . Con las trazas analizadas no se muestra mejora si no hay ningún tipo de anomalía presente en la traza, ya que en general el valor de la entropía no varía a lo largo de toda la traza capturada y esta está muy correlada con el valor de las features originales.

Respecto a la evaluación en presencia de anomalías, en los dos primeros casos las nuevas features tampoco han aportado ninguna mejora significativa por el mismo motivo que en el caso anterior, las features basadas en la entropía también estaban muy correladas con las features originales. Sin embargo, el caso planteado en la tercera anomalía se ha obtenido que las nuevas features son capaces de mejorar mucho los resultados obtenidos, pasando de tener un sistema incapaz de predecir el consumo de CPU de los módulos a un sistema con un error de predicción inferior al 5%

Por otra parte, el cálculo de las nuevas features ha resultado ser muy costo en cuanto a consumo de CPU comparado con las features originales. En este sentido, al añadir las nuevas features se ha incrementado el consumo del sistema un 40% respecto al uso de las features originales, pero estos resultados podrían mejorarse si se optimizase el código de los algoritmos de estimación de entropía y su uso dentro de la herramienta de monitorización CoMo.

En cuanto a trabajos futuros, sería interesante profundizar en el análisis de situaciones en las que la entropía pueda ser un factor determinante para el rendimiento del módulo de Load Shedding. Otra posible línea en la que trabajar es la posibilidad de sustituir algunas de las features originales por las nuevas basadas en la entropía. Esto permitiría cubrir las situaciones en las que las features actuales no sean útiles y reducir al mismo tiempo el coste del cálculo de todo el conjunto de features.

Capítulo 8

Bibliografía

1. **Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun (Jim) Xu and Hui Zhang.** *Data Streaming Algorithms for Estimating Entropy of Network Traffic.* s.l. : SIGMETRICS Performance'06, 2006.

2. **Claude Shannon.** A Mathematical Theory of Communication. [En línea] 1948. http://iie.fing.edu.uy/ense/asign/codif/material/transparencias/01_teoria_de_la_informacion.pdf.

3. **N. Alon, Y. Matias, and M. Szegedy.** *The spacecomplexity of approximating the frequency moments.* s.l. : Proceedings of ACM Symposium on Theory of Computing (STOC), 1996.

4. Waikato Internet Traffic Storage. [En línea] http://www.wand.net.nz/.

5. **Daniela Brauckhoff, Arno Wagner, Martin May.** *FLAME: A Flow-Level Anomaly Modeling Engine.* s.l. : http://www.usenix.org/event/cset08/tech/full_papers/brauckhoff/brauckhoff_html/.

6. Introduction to Cisco IOS NetFlow - A Technical Overview. [En línea] http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.html.

7. Python documentation. [En línea] http://docs.python.org/.

8. Libpcap File Format. [En línea] http://wiki.wireshark.org/Development/LibpcapFileFormat.

9. **Wireshark.** Libpcap File Format. [En línea] http://wiki.wireshark.org/Development/LibpcapFileFormat.

10. **Intel Research Cambridge, UPC, Cambridge University.** The CoMo Project. [En línea] http://como.sourceforge.net/.

11. Snort: A free lightweight network intrusion detection system for UNIX and Windows. [En línea] http://www.snort.org/.

12. **Anukool Lakhina, Mark Crovell and Christophe Diot.** *Characterization of network-wide anomalies in traffic flows.* Portland : SIGCOMM'04 Workshops, 2004.

13. **WAND.** Libtrace Library and ERF Format. [En línea] http://research.wand.net.nz/software/libtrace.php.

14. Boost C++ Libraries. [En línea] http://www.boost.org/.

15. **Gianluca Iannaccone, Christophe Diot, Derek McAuley, Andrew Moore, Ian Pratt, Luigi Rizzo.** The CoMo White Paper. [En línea] 2004. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.872&rep=rep1&type=pdf.

16. **K. Keys, D. Moore and C. Stan.** *A robust system for accurate real-time summaries of internet traffic.* s.l. : Proc. of ACM Sigmetrics, 2005.

17. **D. Carney et al.** *Monitoring streams - a new class of data management applications.* s.l. : Proc. of International Conference on Very Large Data Bases, 2002.

18. **S. Chandrasekaran et al.** *Telegraph CQ: Continuos dataflow processing of an uncertain world.* s.l. : Proc. of Conference on Inovative Data Systems Research, 2003.

19. **P. Barlet-Ros, G. Iannaccone, J. Sanjuas-Cuxart, D. Lopez-Amoros and J. Sole-Pareta.** Predicting resource usage of arbitrary network traffic queries. [En línea] 2006. http://loadshedding.ccaba.upc.edu/prediction.pdf.

20. **Barlet-Ros, Pere, Iannaccone, Gianluca y Josep Sanjuàs-Cuxart, Diego Amores-López, and Josep Solé-Pareta.** *Load Shedding in Network Monitoring Applications.* s.l. : USENIX Annual Technical Conference, 2007.

21. **Liu, L. Yu and H.** *Feature selection for high dimensional data: A fast correlation-based filter solution.* s.l. : Proceedings of ICML, 2003.

22. **Wikipedia.** List of TCP and UDP port numbers. [En línea] http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers.

23. **C. Estan, G. Varghese and M. Fisk.** *Bitmap algorithms for counting active flows on high-speed links.* s.l. : Proc. of ACM Sigcomm Conf. on Internet Measurement, 2003.

24. DPKT. [En línea] http://code.google.com/p/dpkt/.

Capítulo 9

Índice de figuras

Figura 3-1 Propiedades de la Traza 1 22
Figura 3-2 Propiedades de la Traza 2 22
Figura 3-3 Error relativo en la estimación de la entropía del algoritmo de Streaming 24
Figura 3-4 Error relativo en la estimación de la entropía del algoritmo de Sieving 25
Figura 3-5 Eficiencia media del algoritmo de Streaming respecto al cálculo exacto 26
Figura 3-6 Relación entre el número de paquetes y el número de contadores para el algoritmo de Streaming 27
Figura 3-7 Eficiencia media del algoritmo de Sieving y respecto el cálculo exacto 28
Figura 3-8 Error relativo y eficiencia del algoritmo de Streaming respecto a la duración del fragmento de captura al realizar la estimación 28
Figura 3-9 Porcentaje de ciclos de CPU para la estimación de la entropía - Streaming 29
Figura 3-10 Porcentaje de ciclos de CPU para la estimación de la entropía - Sieving 30
Figura 4-1 Proceso de evaluación de una herramienta de gestión de redes 34
Figura 4-2 Contenido de un fichero en formato Libpcap 36
Figura 4-3 Cabecera global de Libpcap 36
Figura 4-4 Cabecera de paquete Libpcap 37
Figura 4-5 Esquema de funcionamiento de recogida de información en Netflow 39
Figura 4-6 Cabecera ERF 40
Figura 4-7 Estructura de generación de una anomalía con FLAME 42
Figura 4-8 Estructura de la herramienta de generación de anomalías 43
Figura 5-1 Estructura de la herramienta CoMo 51
Figura 5-2 Interfaz gráfica CoMo-Live! 53
Figura 5-3 Funcionamiento del módulo de gestión de recursos de CoMo 55
Figura 5-4 Error del predictor de Load Shedding- Features basadas en la entropía 57
Figura 5-5 Error de predicción en función de las features utilizadas - flowcount 58
Figura 5-6 Media y desviación estándar del error de predicción de los 5 módulos 59

Figura 5-7 Porcentaje de aumento de consumo de Load Shedding con las nuevas features 60
Figura 5-8 Entropía de las direcciones origen, destino y puertos origen y destino.....................61
Figura 5-9 Entropía de las direcciones origen, destino y puertos origen y destino.....................63
Figura 5-10 Error de predicción en presencia de un ataque DDoS con 1 módulo activo............63
Figura 5-11 Error de predicción en presencia de un ataque DDoS con 5 módulos activos........64
Figura 5-12 Entropía de las direcciones origen, destino y puertos origen y destino...................65
Figura 5-13 Error de predicción en presencia de un ataque NetScan con 1 módulo activo 66
Figura 5-14 Error de predicción en presencia de un ataque Netscan con 5 módulos activos 66
Figura 5-15 Valor de varias features durante la nueva anomalía...67
Figura 5-16 Error de predicción en presencia de la anomalía: módulo tuple modificado...........68
Figura 6-1 Planificación tareas a realizar 73

Capítulo 10

Índice de tablas

Tabla 3-1 Parámetros de simulación del algoritmo de Streming 23

Tabla 3-2 Parámetros de simulación del algoritmo de Sieving 23

Tabla 5-1 Módulos activos para la evaluación 56

Tabla 5-2 Nuevas features 56

Tabla 6-1 Presupuesto de recursos humanos 74

Tabla 6-2 Coste del equipamiento informático 74

Anexo A

Algoritmos de estimación de la entropía

A.1 Streaming Algorithm

1: **Fase de preprocesado**

2: $z = 32 \log\lceil m/\varepsilon^2\rceil$, $g = 2 \log (1/\delta)$

3: Escoger $z * g$ posiciones en el conjunto de paquetes de forma aleatoria

4: **Fase en tiempo real**

5: Por cada paquete a_j del conjunto hacer

6: si a_j ya tiene uno o más contadores entonces incrementar todos los contadores de a_j

7: si j es una de las posiciones escogidas de forma aleatoria entonces empezar a mantener un contador

8: para a_j inicializado a 1

9: **Fase de post procesado**

10: Considerando los z*g contadores como una matriz C de tamaño $g\times z$

11: para i = 1 hasta g

12: para j = 1 hasta z

13: $X_{i,j} = m * (c_{i,j} \log c_{i,j} - (c_{i,j} - 1) \log (c_{i,j} - 1))$

14: para i = 1 hasta g

15: avg[i] = media de los X_s en los grupos i

16: Devolver la mediana de avg[1], . . . , avg[g]

A.2 Sieving Algorithm

1: **Fase en tiempo real**

2: Por cada paquete del conjunto

3: si el paquete es muestreado entonces

4: si el paquete ya esta siendo contado entonces

5 cambiar la categoría a elefante

6: sino

7: reservar espacio para un contador para este paquete

8: sino

9: incrementar el contador para este paquete si exste

10: **Fase de post procesado**

11: Se = 0

12: Por cada elefante (con la estimación de de su contador c) hacer

13: $S_e = S_e + c \log c$

14: Estimar la contribución de los ratones S_m de los restantes contadores con el algoritmo de Streaming

15: Devolver $S = S_e + S_m$

Anexo B

Modelos de anomalías de ejemplo

A continuación se presentan las tres clases en Python correspondientes a los modelos de anomalías utilizados en el capítulo 6: Denegación de servicio, escaneo de puertos y simulación de valores máximos y mínimos de la entropía.

B.1 DDoS

```
# -*- coding: utf-8 -*-
# Model for Distributed Denial of Service (DDoS) attack
# Injects flow records with random source port and destination port for max_packets
# Usage: Modify addrDotted, dstAddrDotted, startHuRead, endHuRead

import random
import socket,struct
import time
import datetime
import cPickle
import string

class PythonPcapGenerator(PcapGenerator):

  # INPUT PARAMETERS
  # Ethernet Header
  ether_dhost = 0x0030dab84ab5
  ether_shost = 0x001de071877d
  ether_type = 0x0800

  # IP Header
  ip_vhl = 0x45
  ip_tos = 0
  ip_len = 40
  ip_id = random.randint(0,65535)
  ip_off = 0
  ip_ttl = 64
  ip_p = 6
  str_ip_src = "17.43.43.19"
  str_ip_dst = "17.43.43.100"
  ip_src = struct.unpack('!L',socket.inet_aton(str_ip_src))[0]
  ip_dst = struct.unpack('!L',socket.inet_aton(str_ip_dst))[0]
  ip_src_original = ip_src

  # TCP Header
  source = random.randint(3000,65535)
  dest = 80
  seqSrc = random.randint(0,4294967295)
  seqDst = random.randint(0,4294967295)
  ack_seq = 0
  doff = 5  # TCP HEADER LENGTH (length = doff * 4, i.e. 20 bytes)
```

```
  tcp_flags = 0x02 #TCP_SYN
  window = 65535
  urg_ptr = 0

  # Time related parameters
  timestamp = "2009-01-15 13:59:30.000000"

  #Timestamp conversion to seconds and microseconds
  sec = long(time.mktime(time.strptime(timestamp.split('.')[0],'%Y-%m-%d %H:%M:%S')))
  usec = long(timestamp.split('.')[1])

  # Anomaly parameters
  num_packets = 0
  max_packets = 4000000
  usec_increment = 2
  done = False

  # Send request and reply
  def getRequest(self):
    if self.num_packets < self.max_packets:
      self.syn()
      self.sendPacketSrcToDst()
      self.num_packets += 1
      if self.num_packets < 1000000:
          self.synAck()
          self.sendPacketDstToSrc()
      elif (self.num_packets < 2000000) & (random.random() < 0.5):
          self.synAck()
          self.sendPacketDstToSrc()
    elif self.num_packets >= self.max_packets:
      self.done = True

  def getReply(self):
    if self.done == True:
      self.stop()

  # Get properties for next flow
  def syn(self):
    self.ip_src += random.randint(-65535,65535)
    if self.ip_src >= pow(2,32):
      self.ip_src = self.ip_src_original
    self.source = random.randint(3000,65535)
    self.tcp_flags = 0x02
    self.seqSrc = random.randint(0,4294967295)
    self.seqDst = random.randint(0,4294967295)
    self.ack_seq = 0
    self.ip_len = 40
    self.ip_id = random.randint(0,65535)
    self.usec += self.usec_increment
    if self.usec >= 1000000:
      self.sec += 1
      self.usec = self.usec % 1000000

  def synAck (self):
    self.tcp_flags = 0x12
    self.ack_seq = self.seqSrc + 1
    self.ip_len = 40
    self.ip_id = random.randint(0,65535)
    self.usec += self.usec_increment
    if self.usec >= 1000000:
      self.sec += 1
      self.usec = self.usec % 1000000

  def sendPacketSrcToDst(self):
    if(self.ip_p == 6):
      self.sendTcp(self.ether_shost,    self.ether_dhost,    self.ether_type,self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_src,
self.ip_dst,    self.source,    self.dest,    self.seqSrc,    self.ack_seq,    self.doff,
self.tcp_flags, self.window, self.urg_ptr, self.sec, self.usec)
    else:
      self.sendUdp(self.ether_shost,  self.ether_dhost,  self.ether_type,  self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_src,
self.ip_dst, self.source, self.dest, self.len, self.sec, self.usec)

  def sendPacketDstToSrc(self):
    if(self.ip_p == 6):
```

```
      self.sendTcp(self.ether_dhost,    self.ether_shost,    self.ether_type,self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_dst,
self.ip_src,    self.dest,    self.source,    self.seqDst,    self.ack_seq,    self.doff,
self.tcp_flags, self.window, self.urg_ptr, self.sec, self.usec)
    else:
      self.sendUdp(self.ether_dhost,  _self.ether_shost,  self.ether_type,  self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_dst,
self.ip_src, self.dest, self.src, self.len, self.sec, self.usec)
```

B.2 NetScan

```
# Model for port scan attack
# Injects packet records with a small set of destination IP with a defined
# destination port and from a random source.
# Usage:Modify INPUT PARAMETERS related to the Ethernet/IP/TCP headers
# and anomaly parameters

import random
import socket,struct
import time
import datetime
import cPickle
import string

class PythonPcapGenerator(PcapGenerator):

### INPUT PARAMETERS
  # Ethernet Header
  ether_dhost = 0x0030dab84ab5
  ether_shost = 0x001de071877d
  ether_type = 0x0800

  # IP Header
  ip_vhl = 0x45
  ip_tos = 0
  ip_len = 40
  ip_id = random.randint(0,65535)
  ip_off = 0
  ip_ttl = 64
  ip_p = 6
  str_ip_src = "10.0.0.1"
  str_ip_dst = "192.168.1.1"
  ip_src = struct.unpack('!L',socket.inet_aton(str_ip_src))[0]
  ip_dst = struct.unpack('!L',socket.inet_aton(str_ip_dst))[0]

  # TCP Header
  source = random.randint(10000,65535)
  dest = 1
  seq = random.randint(0,4294967295)
  seq2 = random.randint(0,4294967295)
  ack_seq = 0
  doff = 5  # TCP HEADER LENGTH (length = doff * 4, i.e. 20 bytes)
  tcp_flags = 0x02
  window = 65535
  urg_ptr = 0

  # Time related parameters
  timestamp = "2009-11-08 23:44:01.133000"

  #Timestamp converion to seconds and microseconds
  sec = int(time.mktime(time.strptime(timestamp.split('.')[0],'%Y-%m-%d %H:%M:%S')))
  usec = int(timestamp.split('.')[1])

  # Anomaly parameters
  str_ip_dst_end = "192.168.1.255"
  ip_dst_end = struct.unpack('!L',socket.inet_aton(str_ip_dst_end))[0]
  done = False

### Anomaly algorithm
  # Definition of getRequest and getReply functions
  def getRequest(self):
    if self.ip_dst < self.ip_dst_end:
      self.packet1();
      self.sendPacketSrcToDst();
      self.packet2();
      self.sendPacketDstToSrc();
      self.next();
    elif self.ip_dst >= self.ip_dst_end:
      self.done = True

  def getReply(self):
    if self.done == True:
      self.stop()
```

```
### Configuration of portscan
  def packet1(self):
    self.tcp_flags = 0x02
    self.seqSrc = random.randint(0,4294967295)
    self.seqDst = random.randint(0,4294967295)
    self.ack_seq = 0
    self.ip_len = 40
    self.ip_id = random.randint(0,65535)
    self.usec += random.randint(20, 3000)
    if self.usec >= 1000000:
      self.sec += 1
      self.usec = self.usec % 1000000

  def packet2(self):
    self.tcp_flags = 0x12
    self.ack_seq = self.seqSrc + 1
    self.ip_len = 40
    self.ip_id = random.randint(0,65535)
    self.usec += random.randint(20, 3000)
    if self.usec >= 1000000:
      self.sec += 1
      self.usec = self.usec % 1000000

  def next(self):
    self.dest = self.dest + 1
    if self.dest == 65535:
      self.dest = 1
      self.ip_dst = self.ip_dst + 1

### These two functions help to the cleanlyness of the Python code
  def sendPacketSrcToDst(self):
    if(self.ip_p == 6):
      self.sendTcp(self.ether_shost,    self.ether_dhost,    self.ether_type,self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_src,
self.ip_dst,    self.source,    self.dest,    self.seqSrc,    self.ack_seq,    self.doff,
self.tcp_flags, self.window, self.urg_ptr, self.sec, self.usec)
    else:
      self.sendUdp(self.ether_shost,  _self.ether_dhost,  self.ether_type,  self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_src,
self.ip_dst, self.source, self.dest, self.len, self.sec, self.usec)

  # Unmask sendTCP/UDP from dst to src
  def sendPacketDstToSrc(self):
    if(self.ip_p == 6):
      self.sendTcp(self.ether_dhost,    self.ether_shost,    self.ether_type,self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_dst,
self.ip_src,    self.dest,    self.source,    self.seqDst,    self.ack_seq,    self.doff,
self.tcp_flags, self.window, self.urg_ptr, self.sec, self.usec)
    else:
      self.sendUdp(self.ether_dhost,  _self.ether_shost,  self.ether_type,  self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_dst,
self.ip_src, self.dest, self.src, self.len, self.sec, self.usec)
```

B.3 Mínimo y máximo de entropía

```
# Model for minimum flow entropy (1 flow 15360 paquetes, 1023 flows 1 paquete) &
# Model for maximun entropy (1024 flows 16 packets)
# Usage: Modify addrDotted, dstAddrDotted, startHuRead, endHuRead


import random
import socket,struct
import time
import datetime
import cPickle
import string

class PythonPcapGenerator(PcapGenerator):

  # INPUT PARAMETERS
  # Ethernet Header
  ether_dhost = 0x0030dab84ab5
  ether_shost = 0x001de071877d
  ether_type = 0x0800

  # IP Header
  ip_vhl = 0x45
  ip_tos = 0
  ip_len = 40
  ip_id = random.randint(0,65535)
  ip_off = 0
  ip_ttl = 64
  ip_p = 6
  str_ip_src = "17.43.43.19"
  str_ip_dst = "17.43.43.100"
  ip_src = struct.unpack('!L',socket.inet_aton(str_ip_src))[0]
  ip_dst = struct.unpack('!L',socket.inet_aton(str_ip_dst))[0]
  ip_src_original = ip_src

  # TCP Header
  source = random.randint(3000,65535)
  dest = random.randint(3000,65535)
  seqSrc = random.randint(0,4294967295)
  seqDst = random.randint(0,4294967295)
  ack_seq = 0
  doff = 5  # TCP HEADER LENGTH (length = doff * 4, i.e. 20 bytes)
  tcp_flags = 0x02 #TCP_SYN
  window = 65535
  urg_ptr = 0

  # Time related parameters
  timestamp = "2010-09-01 19:05:00.000000"

  #Timestamp conversion to seconds and microseconds
  sec = long(time.mktime(time.strptime(timestamp.split('.')[0],'%Y-%m-%d %H:%M:%S')))
  usec = long(timestamp.split('.')[1])

  # Anomaly parameters
  length_anomaly1 = 0
  num_packets1 = 0
  num_flows1 = 0
  max_flows1 = 1024
  max_packets1 = 16384-max_flows1
  packets_batch1 = 16384
  usec_increment = 6
  empty_secs1 = 100000 - usec_increment * packets_batch1

  length_anomaly2 = 0
  num_packets2 = 0
  num_flows2 = 0
  max_flows2= 16384/16 # max_flows1
  max_packets2 = 16
  packets_batch2 = 16384
  usec_increment = 6
  empty_secs2 = 100000 - usec_increment * packets_batch2
```

```
num_anomalies = 0
max_anomalies = 10
done = False
minimumDone = False
maximumDone = False

# Send request and reply
def getRequest(self):
  if self.num_anomalies < self.max_anomalies:
    if self.minimumDone == False:
      # Minimum Entropy
      if self.num_packets1 < self.max_packets1:
        self.sendPacketSrcToDst()
        self.increaseTime()
        self.num_packets1 += 1
      elif self.num_flows1 < self.max_flows1:
        self.num_flows1 += 1
        self.changeFlow()
        self.sendPacketSrcToDst()
        self.increaseTime()
      elif self.num_flows1 == self.max_flows1:
        #print "Anomaly 1 finished"
        if self.length_anomaly1 >= 10:
          self.length_anomaly1 = 0
          # Finish 5 seconds minimum anomaly
          self.minimumDone = True
        else:
          print self.length_anomaly1
          self.length_anomaly1 += 1
          self.num_packets1 = 0
          self.num_flows1 = 0
          self.usec += self.empty_secs1
          if self.usec >= 1000000:
            self.sec += 1
            self.usec = self.usec % 1000000
          self.changeFlow()
    elif self.maximumDone == False:
      # Maximum entropy
      if self.num_flows2 < self.max_flows2:
        if self.num_packets2 < self.max_packets2:
          self.sendPacketSrcToDst()
          self.increaseTime()
          self.num_packets2 += 1
        elif self.num_packets2 == self.max_packets2:
          self.changeFlow()
          self.num_packets2 = 0
          self.num_flows2 += 1
      elif self.num_flows2 == self.max_flows2:
        if self.length_anomaly2 >= 10:
          self.maximumDone = True
          self.length_anomaly2 = 0
        else:
          print self.length_anomaly2
          self.num_packets2 = 0
          self.num_flows2 = 0
          self.length_anomaly2 += 1
          self.usec += self.empty_secs2
          if self.usec >= 1000000:
            self.sec += 1
            self.usec = self.usec % 1000000
          self.changeFlow()
    else:
      print "anomalia acabada"
      self.num_anomalies += 1
      self.num_packets1 = 0
      self.num_flows1 = 0
      self.minimumDone = False
      self.lengt_anomaly1 = 0
      self.num_packets2 = 0
      self.num_flows2 = 0
      self.maximumDone = False
      self.lengt_anomaly2 = 0
  else:
    self.done = True

def getReply(self):
  if self.done == True:
```

```
      self.stop()

  # Get properties for next flow
  def changeFlow(self):
    self.ip_src += 1
    self.ip_dst += 1

  def increaseTime(self):
    self.usec += self.usec_increment
    if self.usec >= 1000000:
      self.sec += 1
      self.usec = self.usec % 1000000

  def resetAnomalies(self):
    self.num_packets1 = 0
    self.num_flows1 = 0
    self.minimumDone = False
    self.lengt_anomaly1 = 0
    self.num_packets2 = 0
    self.num_flows2 = 0
    self.maximumDone = False
    self.lengt_anomaly2 = 0

  def sendPacketSrcToDst(self):
    if(self.ip_p == 6):
      self.sendTcp(self.ether_shost,    self.ether_dhost,    self.ether_type,self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_src,
self.ip_dst,    self.source,    self.dest,    self.seqSrc,    self.ack_seq,    self.doff,
self.tcp_flags, self.window, self.urg_ptr, self.sec, self.usec)
    else:
      self.sendUdp(self.ether_shost,  _self.ether_dhost,  self.ether_type,  self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_src,
self.ip_dst, self.source, self.dest, self.len, self.sec, self.usec)

  def sendPacketDstToSrc(self):
    if(self.ip_p == 6):
      self.sendTcp(self.ether_dhost,    self.ether_shost,    self.ether_type,self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_dst,
self.ip_src,    self.dest,    self.source,    self.seqDst,    self.ack_seq,    self.doff,
self.tcp_flags, self.window, self.urg_ptr, self.sec, self.usec)
    else:
      self.sendUdp(self.ether_dhost,  _self.ether_shost,  self.ether_type,  self.ip_vhl,
self.ip_tos, self.ip_len, self.ip_id, self.ip_off, self.ip_ttl, self.ip_p, self.ip_dst,
self.ip_src, self.dest, self.src, self.len, self.sec, self.usec)
```

UPC
telecom BCN